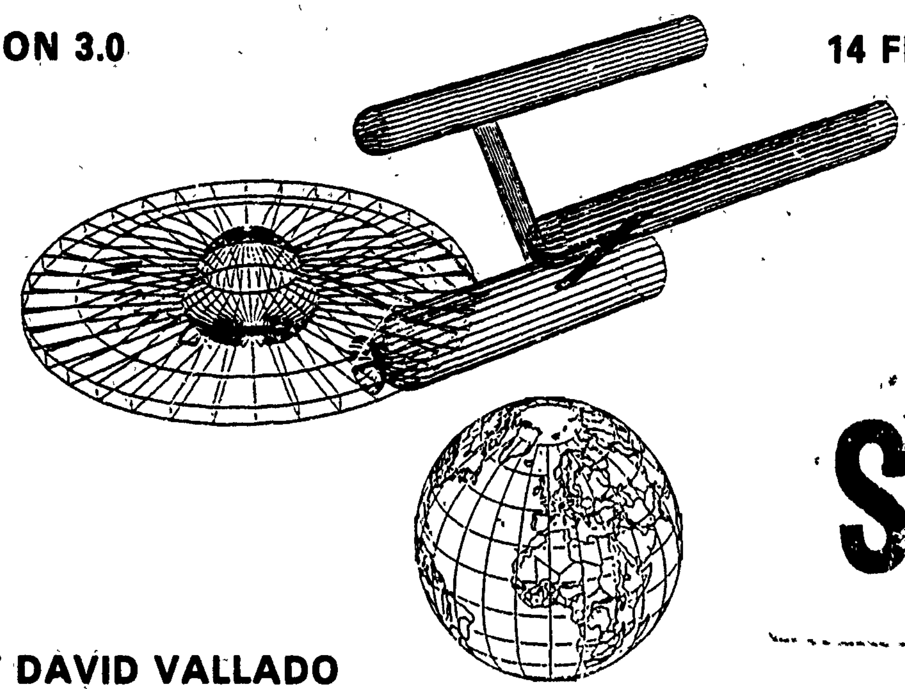
(2)

USAFA-TR-91-6

METHODS OF ASTRODYNAMICS
A COMPUTER APPROACH

VERSION 3.0

14 FEBRUARY 1991



BY CAPT DAVID VALLADO

DEPARTMENT OF ASTRONAUTICS
HEADQUARTERS U.S. AIR FORCE ACADEMY

DEAN OF THE FACULTY

UNITED STATES AIR FORCE ACADEMY

COLORADO 80840

USAFA-TR-91-6

Technical Review by Capt Robert J. Kaufman
Department of Computer Science
USAFA Academy, Colorado 80840

Technical Review by Major David J. Cloud
Department of Astronautics
USAF Academy, Colorado 80840

Editorial Review by Lt Col Donald C. Anderson
Department of English
USAF Academy, Colorado 80840

This research report entitled "Methods of Astrodynamics A Computer Approach" is presented as a competent treatment of the subject, worthy of publication. The United States Air Force Academy vouches for the quality of the research, without necessarily endorsing the opinions and conclusions of the author.

This report has been cleared for open publication and public release by the appropriate Office of Information in accordance with AFM 190-1, AFR 12-30, and AFR 80-3. This report may have unlimited distribution.

Robert K. Morrow, Jr.
ROBERT K. MORROW JR., Lt Col, USAF
Director of Research

25 JUN 91
Dated

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	14 February 1991	Final Report - May 88 to Feb 91

4. TITLE AND SUBTITLE

Methods of Astrodynamics, A Computer Approach

5. FUNDING NUMBERS

6. AUTHOR(S)

Capt David A. Vallado

DSN 8-259-~~4010~~ 4110 or 4109

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

DEPARTMENT OF THE AIR FORCE
HQ USAFA/DFAS
USAF ACADEMY, CO 80840-5701

8. PERFORMING ORGANIZATION REPORT NUMBER

USAFA-TR-91-6

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

None

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

N/A

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A library of PASCAL and FORTRAN computer routines to solve various Astrodynamic problems is presented. Diagrams and equations are given for each routine. The main part of the document is the actual code. Rigorous documentation and coding discipline was used during development of these routines. The code contains extensive information for each routine defining Input / Output variables, local variables, constants, coupling, and references. Finally, test cases and answers are presented.

References are given, therefore formulas are not derived. The code presented will run under TURBO PASCAL Ver5.0 +, MICROSOFT FORTRAN Ver 4.0 +, and VAX FORTRAN Ver 4.6. Both langauges have two files, astrodynamic and mathematical. The PASCAL mathematical code includes a number of mathematical operations not inherent to the langauge. The FORTRAN code was developed in LAHEY FORTRAN Ver 3.0 and uses several features of FORTRAN 90.

Driver programs are not included since these routines are designed to be a library which one may attach to a main program. These routines were developed for academic use and as such, may not match operational data. Information concerning the latest software version may be obtained from the author.

14. SUBJECT TERMS

Astrodynamics	PASCAL	Software Library
Celestial Mechanics	FORTRAN	Software Algorithm
Compucer Algorithms	Orbital Mechanics	Orbital Dynamics

15. NUMBER OF PAGES 376

16. PRICE CODE N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

ABSTRACT

This paper is intended to provide computer routines to solve various Astrodynamic problems. The specific content is several PASCAL and FORTRAN source code routines. Version 1.0 evolved from the USAFA Department of Astronautics class libraries. This collection of routines has evolved over the years. As such, many prior instructors deserve a great deal of credit for most of the original translations and formulations.

My involvement began several years ago with a request from AF SPACECOM/XPSY and XPDY to provide a detailed, documented listing of various Astrodynamic routines. I have included both PASCAL and FORTRAN source code listings since although FORTRAN is still the industry "standard" for technical programming, the Defense Departments desire to use ADA will surely be implemented on a wide scale basis in the near future, and PASCAL is very similar to ADA. In addition, PASCAL allows very easy incorporation of graphics.

Perhaps the most difficult part of this process was the development of test cases sufficient to adequately test all the parameters of each procedure. A variety of sources for problems and answers were assembled. Where answers were given, the check was relatively easy. With no answers, other checks between the routines had to be made. The answers given appear reasonable from all indications but should not be considered an absolute guarantee of the correctness of the algorithm.

The paper is divided into several sections. The main section contains diagrams and an explanation of each procedure. (Note I refer to Procedures, Subroutines and Functions as one item in the narrative) The Appendices contain the PASCAL and FORTRAN source code. Care was taken to provide very similar PASCAL and FORTRAN listings, and identical arguments. The source code also contains extensive information for each procedure defining Input / Output variables, local variables, constants, coupling, and references. Finally, the test cases and answers are presented for each procedure.

The user should be aware I have developed these routines for Acedemic use and as such, they may not match operational data exactly. I have endevered to make each routine completly independent, and identified all other necessary procedures.

TABLE of CONTENTS

TABLE of CONTENTS Continued

Technical Routines:

TABLE of CONTENTS Continued

GENERAL INFORMATION

"Methods of Astrodynamics" is designed to provide PASCAL and FORTRAN source code for various Astrodynamic routines. The paper does NOT derive any of the formulas as the referenced texts do this in great detail. The reader should be aware that the variety of sources is necessary to obtain a "complete" understanding of each algorithm. I have used three main texts in the development of these routines. The references are listed in order of their relative completeness in describing each problem.

Bate, Roger R., Mueller, Donald D., and White, Jerry E., **Fundamentals of Astrodynamics**, New York, Dover Publications, 1970.

Escobal, Pedro R., **Methods of Orbit Determination**, John Wiley & Sons, New York 1965, Reprint Edition Kreiger Publishing Co, Malabar FL , 1979

Kaplan, Marshall H., **Modern Spacecraft Dynamics and Coutrol**, New York, John Wiley & Sons, 1976.

Unit systems present a problem in almost every application of these procedures since each problem has different units. For this reason, I have programmed the routines to use canonical units, and radians exclusively. <u>Every</u> calculation uses these units, and any conversions are performed before the procedure is called. An added benefit of canonical units is their relative magnitudes. By acting almost like a scaling factor, the magnitudes of the numbers are reduced to much smaller scalar-like values. In addition, the Gravitational Parameter of the Earth appears in many places in the equations. The use of the canonical system defines the Gravitational Parameter of the Earth as 1.0, which eliminates a great deal of code. Care should be exercised if the procedures are converted to work in a different unit system. Table 1 lists the constants used in these programs. All these variables have been derived from the World Geodetic Survey 1984. This was accomplished using the three "base" values: equatorial radius (ft), rotational velocity of the Earth, and the Gravitational Parameter of the Earth. If it is desired to change these conversions and constants, be sure to update <u>all</u> of the parameters.

Rotational Velocity of the Earth	7.292115×10^{-5} rad/s
Gravitational Parameter of the Earth	3.986005×10^{5} km^3/s^2

Several "standard coordinate systems are used throughout this paper. The Geocentric Equatorial (IJK) system refers to the Earth centered system with the I axis pointing to the Vernal Equinox and the J axis perpendicular in the orbital plane. The K axis is normal to both I and J and points through the North Pole. Next, the Topocentric Horizon (SEZ) system is used for routines simulating radar sites. In this system, the S axis points due South from the site, and the E axis points due East. The Z axis points straight up from the site and is parallel to the position vector. Escobal presents an excellent discussion of the various coordinate system in Chapter 4 of his book.

The computer code was designed to be as compatible as possible between different computers. To this end, I have adopted several features in my code to facilitate any conversion. The code presented will run under TURBO PASCAL Ver5.5, MICROSOFT FORTRAN Ver 5.0 and VAX FORTRAN Ver 4.6.

In PASCAL, I have tried to avoid many of the powerful features of the language such as pointers, records, and variable type structures. I have used pointers to implement all my matrix operations since this allows the user to use almost any size matrix, within memory constraints. The matrix operations are set up to closely resemble "normal" coding. The user is cautioned to DISPOSE (delete) all matricies when no longer used as iterations can caues lots of memory to be uesd. The PASCAL source code was developed on the Zenith 248, DOS 3.xx, using Turbo Pascal Ver5.5. The code uses the EXTENDED type for all REAL variables. This feature of Turbo Pascal Ver5.0 and later, lets the computer emulate a math co-processor, and have 19-20 significant digits without a co-processor.

In FORTRAN, I have included an IMPLICIT NONE declaration in every SUBROUTINE and FUNCTION. This forces you to declare all variables, and should reduce many errors during program writing. The code was first developed in LAHEY FORTRAN Ver 3.0 and uses several of the extensions to be compatible with FORTRAN 90 when it's released. The FORTRAN code contains no EQUIVALENCES, VERY LIMITED GOTOs and no COMMON blocks in the subroutines library. This is designed so each SUBROUTINE could be passed all of the arguments necessary for its operation.

The user is cautioned when trying to compare EXACT numerical results with the answers in this listing. Numerical accuracies of each machine are different. The use of floating point math, double precision variables, different languages, etc., all make minor differences in the answers. If strict numerical accuracy is needed, new answers may vary from the listing, usually in the 5th or 6th decimal place.

Technically, I have designed these routines to be compatible with a variety of orbit types. To this end, the RandV procedure uses "p" (semi-parameter or semi-latus rectum) as it's input. This even allows calculations for parabolic orbits. I have also used Julian Date as the standard time variable between all routines. This simplifies many operations, and allows procedures like Herrick-Gibbs procedures to function across two days if the sightings occur near local midnight.

I have not included any driver programs since these routines are designed to be a library which one may attach to the main program. In pascal, each file contains similar routines, time, orbit determination, etc., and each is set up as a .TPU file (TURBO PASCALs Unit structure) so the code does not have to be compiled each time. Likewise, the FORTRAN routines are organized the same way, and may be included at the linking step in program developement.

Finally, although I have tried to anticipate any singularities and problem areas in the procedures, an exhaustive search is virtually impossible. For this reason, the Department of Astronautics at the USAFA, and I, cannot take responsibility for maintenance and upkeep of these procedures. If problems do occur, please notify

HQ USAFA / DFAS — Com'l (719) 472-4109
Attn: Capt Dave Vallado — Autovon 259-4109
US Air Force Academy, CO. 80840-5701

for possible assistance and documentation/code changes. Finally, I would appreciate notification of enhancements designed around these routines for possible inclusion into future versions of this software.

World Geodetic Survey 1984

NOTE: ALL times are for SIDEREAL time, except as noted.

MEAN EQUATORIAL RADIUS

	r_e , a_e	1.0 DU	=	20925646.325459318 ft	
			=	3963.190591943 miles	
			=	3443.918466523 NM	
*			=	6378.137000000 km	
		1.0 AU	=	149599650.0 km	

TIME

	1.0 TU	=	13.44685108204	Min
		=	806.81106492270	Sec
		=	0.00933809102919444	Days
	θ_{go} 1990	=	100.3836180°	
	θ_{go} 1991	=	100.1449058°	
	θ_{go} 1992	=	99.9061937°	
	θ_{go} 1993	=	100.6531291°	
	θ_{go} 1994	=	100.4144172°	

SPEED

$1.0\ \frac{DU}{TU}$	=	$25936.241129097825\ \frac{ft}{s}$
	=	$7.905366296149\ \frac{km}{s}$

GRAVITATIONAL PARAMETER

	μ	$1.0\ \frac{DU^3}{TU^2}$	=	$14076443812518712.80\ \frac{ft^3}{s^2}$
*			=	$398600.50000000\ \frac{km^3}{s^2}$

EARTH ROTATION

		=	$0.05883359068688786\ \frac{rad}{TU}$
	$\omega_\oplus$	=	$0.25068444793441402\ \frac{deg}{min}$
*		=	$0.00007292115000000\ \frac{rad}{s}$
		=	$6.30038809866574\ \frac{rad}{solar\ day}$

SHAPE

b_e	Semi-Minor Axis	=	6356.752314200 km			
e_e	Eccentricity of Earth	=	0.08181919084260	e_e^2	=	0.00669437999013
f	Flatenning of Earth	=	1.0 / 298.257223563		=	0.003352810664747352
J_2		=	0.00108263			
J_3		=	-0.00000254			
J_4		=	-0.00000161			

* indicates defining parameter for WGS-84

CONVERSIONS

FtToKM	=	0.0003048
NmToKm	=	1.8520
MilesToKm	=	1.6093440
FtToMiles	=	1.0 / 5280.0
$\frac{\pi}{2}$	=	1.57079632679490
π	=	3.14159265358979
2π	=	6.28318530717959
1.0 radian	=	57.29577951308230°
$1.0\ \frac{deg}{s}$	=	$1.0\ /\ 0.0710151137039398\ \frac{rad}{TU}$

REFERENCES

1. Bate, Roger R., Mueller, Donald D., and White, Jerry E., **Fundamentals of Astrodynamics**, New York, Dover Publications, 1970.

2. Battin, Richard H.,**An Introduction to the Mathematics and Methods of Astrodynamics**, AIAA Education Series, Wright Patterson AFB, OH, 1987.

3. Brower, Dirk and Clemence, Gerald M., **Methods of Celestial Mechanics**, Academic Press Inc, New York, 1961.

4. Danby, J.M.A., **Fundamentals of Celestial Mechanics**, Willmann Bell Inc, Richmond VA, 1988.

5. Escobal, Pedro R., **Methods of Orbit Determination**, John Wiley & Sons, New York 1965, Reprint Edition Kreiger Publishing Co, Malabar FL , 1979

6. Escobal, Pedro R., **Methods of Astrodynamics**, John Wiley & Sons, New York 1968, Reprint Edition Kreiger Publishing Co, Huntington New York, 1979

7. Fitzpatrick, Philip M., **Principles of Celestial Mechanics**, New York, Academic Press, 1970.

8. Kaplan, Marshall H., **Modern Spacecraft Dynamics and Control**, New York, John Wiley & Sons, 1976.

9. Moulton, Forest Ray, **An Introduction to Celestial Mechanics**, New York, Dover Publications, 1970.

10. Regan, Frank J., **Re-Entry Vehicle Dynamics**, AIAA Education Series, Wright Patterson AFB OH, 1984.

11. Roy, Archie E., **Orbital Motion**, New York, John Wiley & Sons, 1978.

12. Wertz, James R., **Spacecraft Attitude Determination and Control**, Klower Academic Publishing, Boston, 1988.

13. Szebehely, Victor, **Theory of Orbits, The restricted Problem of Three Bodies**, Academic Press, New York, 1967.

JULIANDAY (Yr,Mon,D,H,M,Sec, JD)

This procedure finds the Julian date given the Year, Month, Day, and Time. The Julian date is defined by each elapsed day since noon, 1 Jan 4713 BC. Julian dates are measured from this epoch at noon so astronomers observations may be performed on a single "day". The year range is limited since machine routines for 365 days a year and leap years are valid in this range only. This is due to the fact that leap years occur only in years divisible by 4 and centuries whose number is evenly divisible by 400. (1900 no, 2000 yes ...)

NOTE: This Algorithm is taken from the 1988 Almanac for Computers, Published by the U.S. Naval Observatory. The algorithm is good for dates between 1 Mar 1900 to 28 Feb 2100 since the last two terms (from the Almanac) are commented out.

	Variable			Range
Inputs :	Yr	-	Year	1900 .. 2100
	Mon	-	Month	1 .. 12
	D	-	Day	1 .. 28,29,30,31
	H	-	Universal Time Hour	0 .. 23
	M	-	Universal Time Min	0 .. 59
	Sec	-	Universal Time Sec	0.0 .. 59.999
Outputs :	JD	-	Julian Date	days from 4713 B.C.

References :

1988 Almanac for Computers pg. B2
Escobal pg. 17-19
Kaplan pg. 329-330

$$\text{JDate} = 367\,(\text{Yr}) - \left[\text{INT}\left(\frac{7\left(\text{Yr} + \text{INT}\left(\frac{\text{Mon}+9}{12}\right)\right)}{4}\right)\right] + \text{INT}\left(\frac{275\ \text{Mon}}{9}\right) + \text{Day} + 1721013.5 + \frac{\left(\frac{\left(\frac{\text{Sec}}{60} + \text{Min}\right)}{60}\right) + \text{Hr}}{24}$$

GSTIME (JD)

This function finds the Greenwich Sidereal time. Notice just the integer part of the Julian Date is used for the Julian centuries calculation.

			Variable	Range
Inputs	: JD	-	Julian Date	days from 4713 B.C.
OutPuts :	GSTime	- GST	Greenwich Sidereal Time	0.0 to 2π rad
Locals	: Tu	-	Julian Centuries from 1 Jan 2000	
Constants :				
	RadPerDay		Radians the earth rotates in 1 sidereal day	6.30038809866574

References :

1988 Almanac for Computers pg. B2
1989 Nautical Almanac pg. B6
Escobal pg. 18 - 21
Kaplan pg. 330-332
BMW pg. 103-104

Two Primary methods.

1. Use Julian Date. - Preferred since one value may be used throughout many years.

$$Tu = \frac{JD - 2451545.0}{36525.0} \qquad \text{(Note use of Epoch : 1 Jan 2000)}$$

$$GST_0 = 1.753368559 + 628.3319705Tu + 6.770708127^{-06}Tu^2 \qquad \text{radians}$$

$$GST_0 = 100.4606184 + 36000.77004Tu + 0.00038793Tu^2 \qquad \text{degrees}$$

$$GST = GST_0 + RadPerDay(\ Fraction(\ JDate\ )\ )$$

2. Use tables for a particular year. Requires knowledge of GST at some Epoch.

Look up value for GST_0 for 1 Jan, 0 Hr of the given year.

$$GST = GST_0 + 1.0027379093\ (2\pi)\ (\ DayofYr + \frac{Hr}{24} + \frac{Min}{1440} + \frac{Sec}{86400}\ )$$

Notice the day of year is really (Day - 1) since the epoch is 1 Jan

Don't forget the result is MODed by 2π since the equations give the radians since the epoch.

LSTIME (Lon,JD,   Lst,Gst)

This procedure finds the Local Sidereal time at a given location.

			Variable	Range
Inputs :	Lon	-	Site longitude (WEST -)	-2π to 2π rad
	JD	-	Julian Date	days from 4713 B.C.
OutPuts :	LST	-	Local Sidereal Time	0.0 to 2π rad
	GST	-	Greenwich Sidereal Time	0.0 to 2π rad

Coupling :

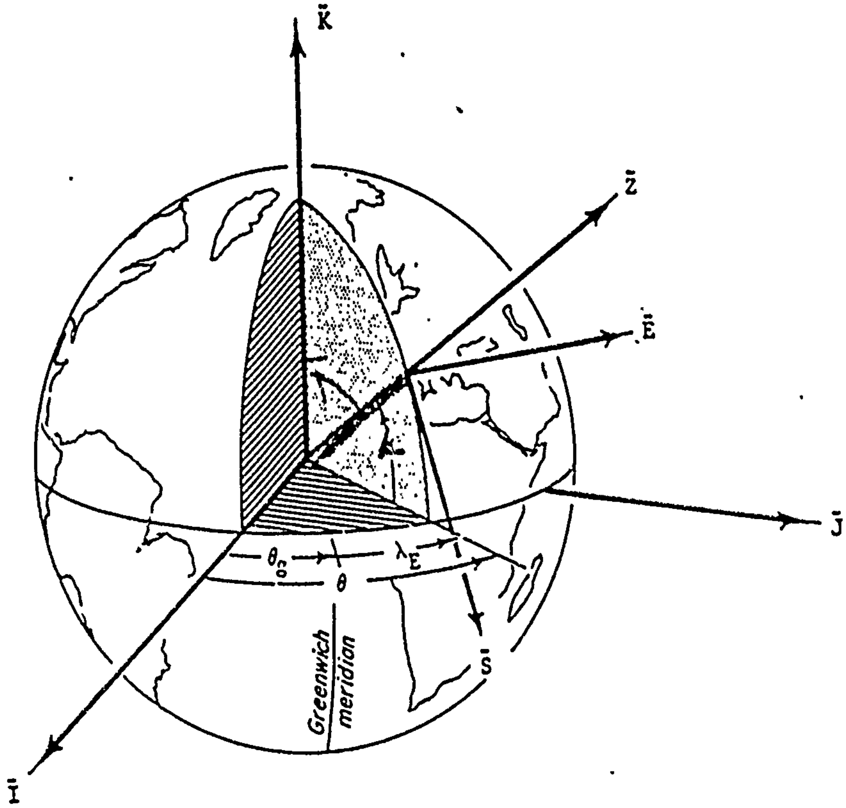
GSTime   Finds the Greenwich Sidereal Time

References :

Escobal	pg. 18 - 21
Kaplan	pg. 330-332
BMW	pg. 99 -100 Diagram pg. 100

Find GST using GSTime procedure (Uses Julian Date)

LST = GST + Lon   (Note: East longitude is + and West lon is –)



SITE (Lat,Alt,LST, RS,VS)

This procedure finds the position and velocity vectors for a site. The answer is returned in the Geocentric Equatorial (IJK) coordinate system.

			Variable	Range
Inputs	:Lat	-	Geodetic Latitude	$-\pi/2$ to $\pi/2$ rad
	Alt	-	Altitude	DUs
	LST	-	Local Sidereal Time	0.0 to 2π rad
OutPuts	:RS	- $\overline{RS}_{ijk}$	Site position vector	DU
	VS	- $\overline{VS}_{ijk}$	Site velocity vector	DU / TU
Locals	:			
	x	-	x component of site vector	DU
	z	-	z component of site vector	DU
Constants	:			
		- a_e	Mean Equatorial Radius of the Earth	1.0 DU
	EESqrd	- e^2_e	Eccentricity of Earths shape squared	0.00669437999013
	OmegaEarth	- $\omega_\oplus$	Angluar Rotation of the Earth	0.0588335906887886 rad/TU

References :

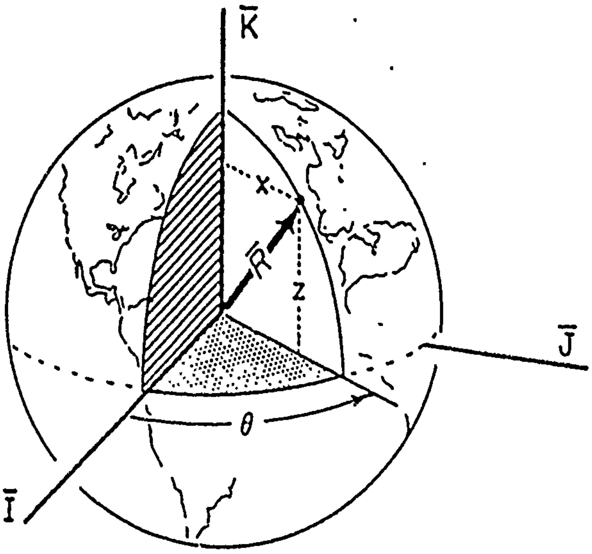
Escobal	pg. 26 - 29 (includes Geocentric Lat formulation also)
Kaplan	pg. 334-336
BMW	pg. 94 - 98 Diagram pg. 99

$$x = \left| \frac{a_e}{\sqrt{1-e_e^2\mathrm{Sin}^2(\,lat\,)}} + alt \right| \mathrm{Cos}(\,lat\,)$$

$$z = \left| \frac{a_e(\,1-e_e^2\,)}{\sqrt{1-e_e^2\mathrm{Sin}^2(\,lat\,)}} + alt \right| \mathrm{Sin}(\,lat\,)$$

$$\overline{RS}_{ijk} = \begin{bmatrix} x\mathrm{Cos}(\,lst\,) \\ x\mathrm{Sin}(\,lst\,) \\ z \end{bmatrix}$$

$$\overline{VS}_{ijk} = \begin{bmatrix} 0 \\ 0 \\ \omega_\oplus \end{bmatrix} \times \overline{RS}_{ijk}$$



RVtoPOS (rho,az,el,drho,daz,del, RhoVec,DRhoVec)

This procedure finds range and velocity vectors for a satellite from a radar site in the Topocentric Horizon (SEZ) system.

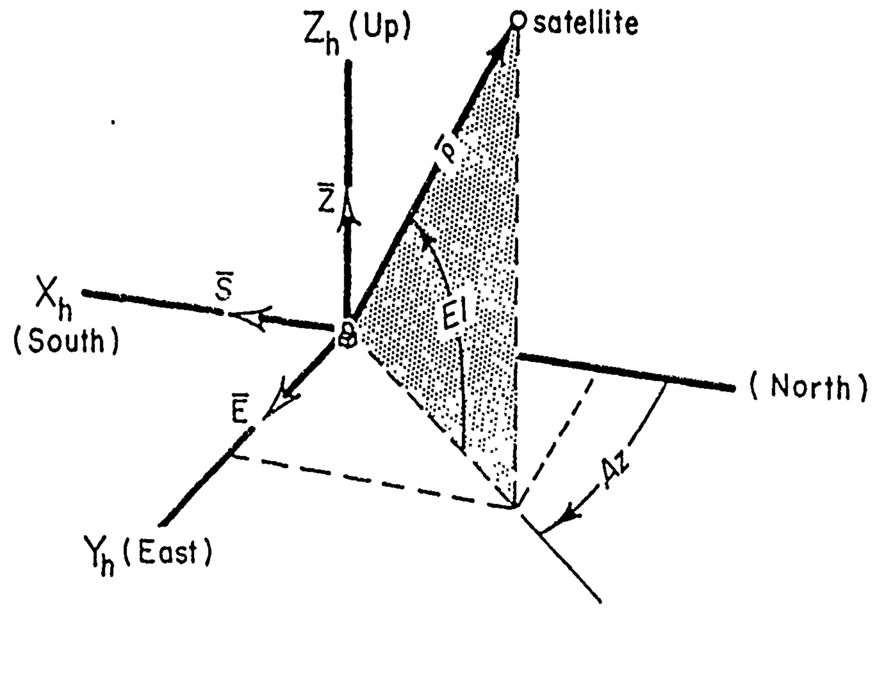
			Variable	Range
Inputs	:Rho	- ρ	Satellite range from site	DUs
	Az	-	Azimuth	0.0 to 2π rad
	El	-	Elevation	$-\pi/2$ to $\pi/2$ rad
	DRho	- $\dot{\rho}$	Range Rate	DU / TU
	DAz	- $A\dot{z}$	Azimuth Rate	rad / TU
	DEl	- $E\dot{l}$	Elevation rate	rad / TU
Outputs	:RhoVec	- $\overline{\rho}_{sez}$	Satellite range vector	DU
	DRhoVec	- $\dot{\overline{\rho}}_{sez}$	Satellite velocity vector	DU / TU

References :

BMW pg. 84 - 85 Diagram pg. 84

$$\overline{\rho}_{sez} = \begin{bmatrix} -\rho \mathrm{Cos}(\, el\,)\, \mathrm{Cos}(\, az\,) \\ \rho\, \mathrm{Cos}(\, el\,)\, \mathrm{Sin}(\, az\,) \\ \rho\, \mathrm{Sin}(\, el\,) \end{bmatrix}$$

$$\dot{\overline{\rho}}_{sez} = \begin{bmatrix} -\dot{\rho}\mathrm{Cos}(\, el\,)\mathrm{Cos}(\, az\,) + \rho \mathrm{Sin}(\, el\,)\mathrm{Cos}(\, az\,)e\dot{l} + \rho \mathrm{Cos}(\, el\,)\mathrm{Sin}(\, az\,)a\dot{z} \\ \dot{\rho}\, \mathrm{Cos}(\, el\,)\mathrm{Sin}(\, az\,) - \rho \mathrm{Sin}(\, el\,)\mathrm{Sin}(\, az\,)e\dot{l} + \rho \mathrm{Cos}(\, el\,)\mathrm{Cos}(\, az\,)a\dot{z} \\ \dot{\rho}\, \mathrm{Sin}(\, el\,) + \rho \mathrm{Cos}(\, el\,)e\dot{l} \end{bmatrix}$$



TRACK (rho,az,el,drho,daz,del,Lat,LST,RS, R,V)

This procedure finds range and velocity vectors in the Geocentric Equatorial (IJK) system given input from a radar site.

			Variable	Range
Inputs	:Rho	- ρ	Satellite range from site	DUs
	Az	-	Azimuth	0.0 to 2π rad
	El	-	Elevation	$-\pi/2$ to $\pi/2$ rad
	DRho	- $\dot{\rho}$	Range Rate	DU / TU
	DAz	- $\dot{Az}$	Azimuth Rate	rad / TU
	DEl	- $\dot{El}$	Elevation rate	rad / TU
	Lat	-	Geodetic Latitude	$-\pi/2$ to $\pi/2$ rad
	LST	-	Local Sidereal Time	0.0 to 2π rad
	RS	- $\overline{RS}_{ijk}$	Site position vector	DU
Outputs	:R	- $\bar{r}_{ijk}$	Satellite position vector	DU
	V	- $\bar{v}_{ijk}$	Satellite velocity vector	DU / TU
Locals	:			
	RhoVec	- $\overline{\rho}_{sez}$	range vector from site	DU
	DRhoVec	- $\dot{\overline{\rho}}_{sez}$	velocity vector from site	DU / TU
	RhoV	- $\overline{\rho}_{ijk}$	range vector from site	DU
	DRhoV	- $\dot{\overline{\rho}}_{ijk}$	velocity vector from site	DU / TU
Constants	:			
	OmegaEarth	ω_o	Angluar Rotation of the Earth	0.0588335906888786 rad/TU

Coupling :

RVToPos — Find R and V from site in Topocentric Horizon (SEZ) system

References :

BMW — pg. 85-89, 100-101 — Diagram pg. 88

Use procedure RVToPOS to Find $\overline{\rho}_{sez}$ and $\dot{\overline{\rho}}_{sez}$

$$\overline{\rho}_{ijk} = \begin{bmatrix} \mathrm{Sin(lat)Cos(lst)} & \mathrm{-Sin(lst)} & \mathrm{Cos(lat)Cos(lst)} \\ \mathrm{Sin(lat)Sin(lst)} & \mathrm{Cos(lst)} & \mathrm{Cos(lat)Sin(lst)} \\ \mathrm{-Cos(lat)} & 0 & \mathrm{Sin(lat)} \end{bmatrix} \overline{\rho}_{sez}$$

$$\bar{r}_{ijk} = \overline{\rho}_{ijk} + \overline{RS}_{ijk}$$

$$\dot{\overline{\rho}}_{ijk} = \begin{bmatrix} \mathrm{Sin(lat)Cos(lst)} & \mathrm{-Sin(lst)} & \mathrm{Cos(lat)Cos(lst)} \\ \mathrm{Sin(lat)Sin(lst)} & \mathrm{Cos(lst)} & \mathrm{Cos(lat)Sin(lst)} \\ \mathrm{-Cos(lat)} & 0 & \mathrm{Sin(lat)} \end{bmatrix} \dot{\overline{\rho}}_{sez}$$

$$\overline{\omega \times r} = \begin{bmatrix} 0 \\ 0 \\ \omega_{\oplus} \end{bmatrix} \times \bar{r}_{ijk}$$

$$\bar{v}_{ijk} = \dot{\overline{\rho}}_{ijk} + \overline{\omega \times r}$$

RAZEL (R,V,Lat,LST,RS, rho,az,el,drho,daz,del)

This procedure calculates Range Azimuth and Elevation and their rates given the Geocentric Equatorial (IJK) Position and Velocity vectors.

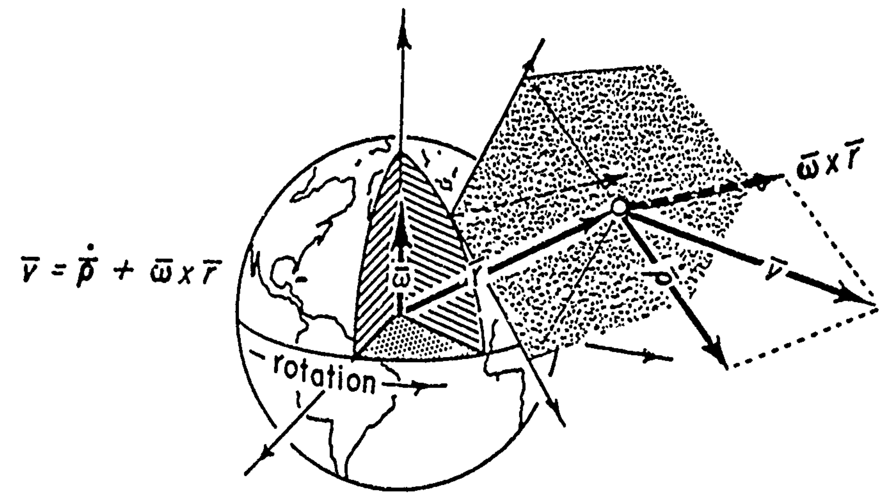
			Variable	Range
Inputs	:R	- $\bar{r}_{ijk}$	Position Vector	DU
	V	- $\bar{v}_{ijk}$	Velocity Vector	DU / TU
	Lat	-	Geodetic Latitude	$-\pi/2$ to $\pi/2$ rad
	LST	-	Local Sidereal Time	0.0 to π rad
	RS	- $\overline{RS}_{ijk}$	Site Position Vector	DU
Outputs	:Rho	- ρ	satellite range from site	DUs
	Az	-	Azimuth	0.0 to 2π rad
	El	-	Elevation	$-\pi/2$ to $\pi/2$ rad
	DRho	- $\dot{\rho}$	Range Rate	DU / TU
	DAz	- $A\dot{z}$	Azimuth Rate	rad / TU
	DEl	- $E\dot{l}$	Elevation rate	rad / TU
Locals	:RhoV	- $\bar{\rho}_{ijk}$	Range Vector from site	DU
	DRhoV	- $\dot{\bar{\rho}}_{ijk}$	Velocity Vector from site	DU / TU
	RhoVec	- $\bar{\rho}_{sez}$	Range vector from site	DU
	DRhoVec	- $\dot{\bar{\rho}}_{sez}$	Velocity vector from site	DU / TU

Constants :

OmegaEarth ω_o	Angluar Rotation of the Earth	0.058833590688786 rad/TU

References :

BMW pg. 84-89, 100-101



$$\bar{\rho}_{ijk} = \bar{r}_{ijk} - \overline{RS}_{ijk}$$

$$\overline{\omega \times r} = \begin{bmatrix} 0 \\ 0 \\ \omega_{\oplus} \end{bmatrix} \times \bar{r}_{ijk}$$

$$\dot{\bar{\rho}}_{ijk} = \dot{\bar{v}}_{ijk} - \overline{\omega \times r}$$

$$\bar{\rho}_{sez} = \begin{bmatrix} \mathrm{Sin(lat)Cos(lst)} & \mathrm{Sin(lat)Sin(lst)} & \mathrm{-Cos(lat)} \\ \mathrm{-Sin(lst)} & \mathrm{Cos(lst)} & 0 \\ \mathrm{Cos(lat)Cos(lst)} & \mathrm{Cos(lat)Sin(lst)} & \mathrm{Sin(lat)} \end{bmatrix} \bar{\rho}_{ijk}$$

$$\dot{\bar{\rho}}_{sez} = \begin{bmatrix} \mathrm{Sin(lat)Cos(lst)} & \mathrm{Sin(lat)Sin(lst)} & \mathrm{-Cos(lat)} \\ \mathrm{-Sin(lst)} & \mathrm{Cos(lst)} & 0 \\ \mathrm{Cos(lat)Cos(lst)} & \mathrm{Cos(lat)Sin(lst)} & \mathrm{Sin(lat)} \end{bmatrix} \dot{\bar{\rho}}_{ijk}$$

$$\mathrm{El} = \mathrm{ATan2}\left(\frac{\rho_z}{\rho}, \frac{\sqrt{\rho_c^2 + \rho_e^2}}{\rho}\right)$$

$$\mathrm{Az} = \mathrm{ATan2}\left(\frac{\rho_e}{\sqrt{\rho_s^2 + \rho_e^2}}, \frac{-\rho_s}{\sqrt{\rho_s^2 + \rho_e^2}}\right)$$

Rate terms are found by rearranging relations in procedure RVTOPOS

$$\dot{\rho} = \frac{\bar{\rho}_{sez} \cdot \dot{\bar{\rho}}_{sez}}{\rho}$$

$$\dot{\mathrm{Az}} = \frac{\dot{\rho}_i \rho_j - \dot{\rho}_j \rho_i}{\rho_i^2 + \rho_j^2}$$

$$\dot{\mathrm{El}} = \frac{\dot{\rho}_k - \dot{\rho}\,\mathrm{Sin}\,(\,\mathrm{El}\,)}{\sqrt{\rho_i^2 + \rho_j^2}}$$

ELORB (R,V, p,a,e,inc,Omega,Argp,Nuo,M,u,l,CapPi)

This procedure finds the classical orbital elements given the Geocentric Equatorial Position and Velocity vectors. Special cases for equatorial and circular orbits are also handled.

			Variable		Range
Inputs	:R	- $\bar{r}$	IJK Position vector		DU
	V	- $\bar{v}$	IJK Velocity vector		DU / TU
Outputs	:p	-	Semi-latus rectum		DU
	a	-	semi-major axis		DU
	e	-	eccentricity		
	inc	- i	inclination		0.0 to π rad
	Omega	- Ω	Longitude of Ascending Node		0.0 to 2π rad
	Argp	- ω	Argument of Perigee		0.0 to 2π rad
	Nuo	- ν	True anomaly		0.0 to 2π rad
	u	-	Argument of Latitude	(CI)	0.0 to 2π rad
	l	-	True Longitude	(CE)	0.0 to 2π rad
	CapPi	- Π	Longitude of Periapsis	(EE)	0.0 to 2π rad
	M	-	Mean Anomaly		0.0 to 2π rad
Locals	:				
	Hbar	- $\bar{h}$	Angular Momentum		DU^2 / TU
	Ebar	- $\bar{e}$	Eccentricity		
	Nbar	- $\bar{n}$	Line of Nodes		
	SME	- $\mathcal{E}$	Specfic Mechanical Energy		DU^2 / TU^2

References :

BMW	pg. 58 - 71
Escobal	pg. 104-107
Kaplan	pg. 29 - 37

$\bar{h} = \bar{r} \times \bar{v}$

$\bar{n} = \hat{k} \times \bar{h}$

$\bar{e} = \frac{1}{\mu}\left[\left(v^2 - \frac{\mu}{r}\right)\bar{r} - (\bar{r} \cdot \bar{v})\bar{v}\right]$

$\mathcal{E} = \frac{v^2}{2} - \frac{\mu}{r}$

$a = -\frac{\mu}{2\mathcal{E}}$

$p = \frac{h^2}{\mu}$

$i = \mathrm{Cos}^{-1}\left[\frac{\hat{k} \cdot \bar{h}}{k\,h}\right]$ i is always between 0.0 and π

$\Omega = \mathrm{Cos}^{-1}\left[\frac{\hat{i} \cdot \bar{n}}{i\,n}\right]$ If $n[j] < 0$ Then $\Omega = 2\pi - \Omega$

$\omega = \mathrm{Cos}^{-1}\left[\frac{\bar{n} \cdot \bar{e}}{n\,e}\right]$ If $e[k] < 0$ Then $\omega = 2\pi - \omega$

$\nu = \mathrm{Cos}^{-1}\left[\frac{\bar{e} \cdot \bar{r}}{e\,r}\right]$ If $\bar{r} \cdot \bar{v} < 0$ Then $\nu = 2\pi - \nu$

Evaluate Special cases
If Circular Inclined:

$u = \mathrm{Cos}^{-1}\left[\frac{\bar{n} \cdot \bar{r}}{n\,r}\right]$ If $r[k] < 0$ Then $u = 2\pi - u$

If Circular Equatorial:

$l = \mathrm{Cos}^{-1}\left[\frac{\hat{i} \cdot \bar{r}}{i\,r}\right]$ If $r[j] < 0$ Then $l = 2\pi - l$ and
If Inc $> \frac{\pi}{2}$ Then $l = 2\pi - l$

IF Elliptical Equatorial:

$\Pi = \mathrm{Cos}^{-1}\left[\frac{\hat{i} \cdot \bar{e}}{i\,e}\right]$ If $e[j] < 0$ Then $\Pi = 2\pi - \Pi$
If Inc $> \frac{\pi}{2}$ Then $\Pi = 2\pi - \Pi$

Find Mean Anomaly
IF Hyperbolic:

$$F = \mathrm{Cosh}^{-1}\left(\frac{e + \mathrm{Cos}\,\nu}{1 + e\,\mathrm{Cos}\,\nu}\right)$$

$$M = e\,\mathrm{Sinh}(F) - F$$

IF Parabolic:

$$D = \sqrt{p}\,\mathrm{Tan}(\nu)$$

$$M = \frac{1}{6}\left(3pD + D^3\right)$$

IF Elliptical:

$$E = \mathrm{ATAN2}\left(\frac{\sqrt{1 - e^2}\,\mathrm{Sin}\,\nu}{1 + e\,\mathrm{Cos}\,\nu},\ \frac{e + \mathrm{Cos}\,\nu}{1 + e\,\mathrm{Cos}\,\nu}\right)$$

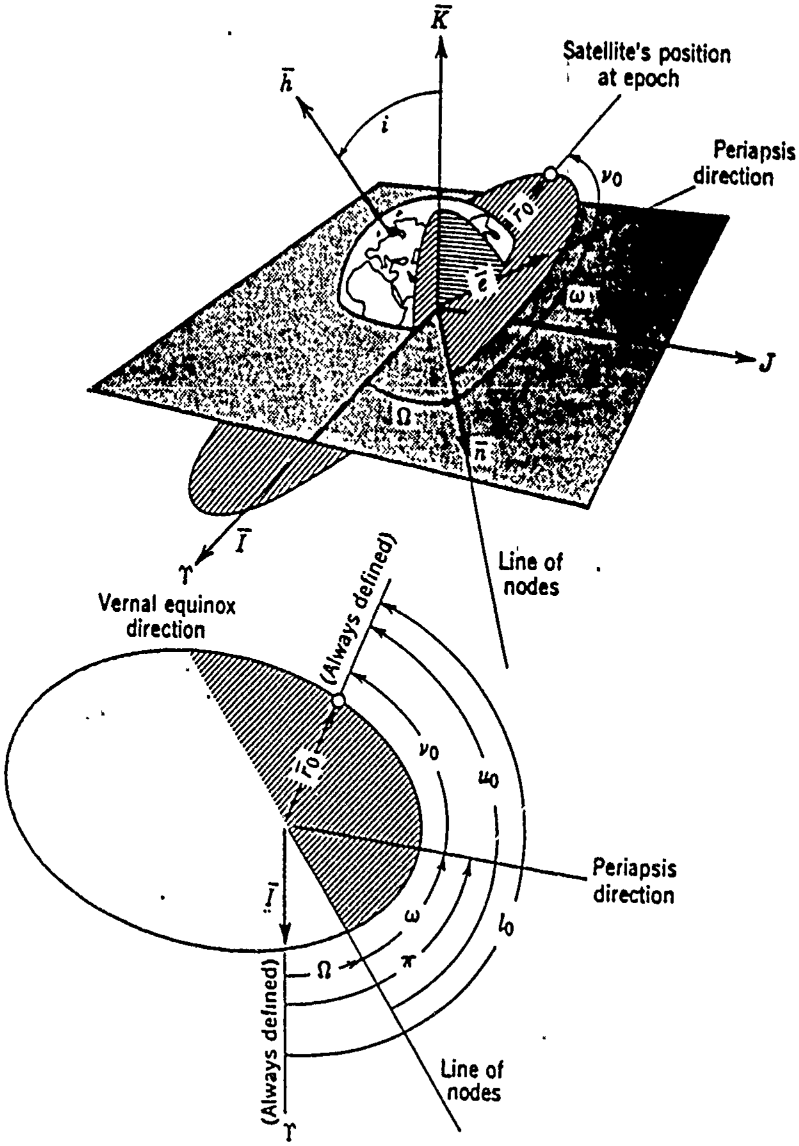
$$M = E - e\,\mathrm{Sin}(E)$$

IF Circular:

M = L if Circular Equatorial
or
M = U if Circular Inclined

Classical Orbit Elements

Ref BMW pg. 59



RANDV (p,a,e,inc,Omega,Argp,Nuo,u,l,CapPi, R,V)

This procedure finds the position and velocity vectors in Geocentric Equatorial (IJK) system given the classical orbit elements. NOTICE P is used for calculations and that A is not used at this time. This convention allows parabolic orbits to be treated as well as the other conic sections.

			Variable		Range
Inputs	: p	-	Semi-latus rectum		DU
	a	-	semi-major axis		DU
	e	-	eccentricity		
	inc	- i	inclination		0.0 to π rad
	Omega	- Ω	Longitude of Ascending Node		0.0 to 2π rad
	Argp	- ω	Argument of Perigee		0.0 to 2π rad
	Nuo	- ν	True anomaly		0.0 to 2π rad
	u	-	Argument of Latitude	(CI)	0.0 to 2π rad
	l	-	True Longitude	(CE)	0.0 to 2π rad
	CapPi	- Π	Longitude of Periapsis	(EE)	0.0 to 2π rad
Outputs	:R	- $\bar{r}_{ijk}$	Position vector		DU
	V	- $\bar{v}_{ijk}$	Velocity vector		DU / TU
Locals	Rpqw	- $\bar{r}_{pqw}$	Position vector		DU
	Vpqw	- $\bar{v}_{pqw}$	Velocity vector		DU / TU

References :

BMW	pg. 71-73, 80-83
Escobal	pg. 68-83

Determine transformation angles for special cases as:

If Circular Equatorial:
set ω, $\Omega = 0.0$ and let $\nu = 1$

If Circular Inclined:
set $\omega = 0.0$ and let $\nu = u$

If Elliptical Equatorial:
set $\Omega = 0.0$ and let $\omega = \Pi$

$$\bar{r}_{pqw} = \begin{bmatrix} \dfrac{p\,\mathrm{Cos}(\nu)}{1 + e\mathrm{Cos}(\nu)} \\ \dfrac{p\,\mathrm{Sin}(\nu)}{1 + e\mathrm{Cos}(\nu)} \\ 0 \end{bmatrix}$$

$$\bar{v}_{pqw} = \begin{bmatrix} \sqrt{\frac{\mu}{p}}\,-\mathrm{Sin}(\nu) \\ \sqrt{\frac{\mu}{p}}\,e + \mathrm{Cos}(\nu) \\ 0 \end{bmatrix}$$

$$\bar{R}_{ijk} = \begin{bmatrix} \mathrm{Cos}\,\Omega\mathrm{Cos}\,\omega - \mathrm{Sin}\,\Omega\mathrm{Sin}\,\omega\mathrm{Cos}\,i & -\mathrm{Cos}\,\Omega\mathrm{Sin}\,\omega - \mathrm{Sin}\,\Omega\mathrm{Cos}\,\omega\mathrm{Cos}\,i & +\mathrm{Sin}\,\Omega\mathrm{Sin}\,i \\ \mathrm{Sin}\,\Omega\mathrm{Cos}\,\omega + \mathrm{Cos}\,\Omega\mathrm{Sin}\,\omega\mathrm{Cos}\,i & -\mathrm{Sin}\,\Omega\mathrm{Sin}\,\omega + \mathrm{Cos}\,\Omega\mathrm{Cos}\,\omega\mathrm{Cos}\,i & -\mathrm{Cos}\,\Omega\mathrm{Sin}\,i \\ \mathrm{Sin}\,\omega\mathrm{Sin}\,i & \mathrm{Cos}\,\omega\mathrm{Sin}\,i & \mathrm{Cos}\,i \end{bmatrix} \bar{R}_{pqw}$$

$$\bar{V}_{ijk} = \begin{bmatrix} \mathrm{Cos}\,\Omega\mathrm{Cos}\,\omega - \mathrm{Sin}\,\Omega\mathrm{Sin}\,\omega\mathrm{Cos}\,i & -\mathrm{Cos}\,\Omega\mathrm{Sin}\,\omega - \mathrm{Sin}\,\Omega\mathrm{Cos}\,\omega\mathrm{Cos}\,i & +\mathrm{Sin}\,\Omega\mathrm{Sin}\,i \\ \mathrm{Sin}\,\Omega\mathrm{Cos}\,\omega + \mathrm{Cos}\,\Omega\mathrm{Sin}\,\omega\mathrm{Cos}\,i & -\mathrm{Sin}\,\Omega\mathrm{Sin}\,\omega + \mathrm{Cos}\,\Omega\mathrm{Cos}\,\omega\mathrm{Cos}\,i & -\mathrm{Cos}\,\Omega\mathrm{Sin}\,i \\ \mathrm{Sin}\,\omega\mathrm{Sin}\,i & \mathrm{Cos}\,\omega\mathrm{Sin}\,i & \mathrm{Cos}\,i \end{bmatrix} \bar{V}_{pqw}$$

GIBBS(r1,r2,r3, v2,Theta)

This procedure performs the Gibbs method of orbit determination. This method determines the velocity at the middle point of the 3 given position vectors. The Gibbs method is best suited for coplanar, sequential position vectors which are more than 10 deg apart. Notice the angle between the vectors is passed back so the user may make a decision about the accuracy of the calculations as vectors which are 120 deg apart may be accurate, while vectors 8 deg apart may not. The method will calculate the resulting velocity using the vectors IN THE ORDER GIVEN.

			Variable	Range
Inputs	:R1	- $\bar{r}_1$	IJK Position vector #1	DU
	R2	- $\bar{r}_2$	IJK Position vector #2	DU
	R3	- $\bar{r}_3$	IJK Position vector #3	DU
Outputs	:V2	- $\bar{v}_2$	Velocity Vector for R2	DU / TU
	Theta	-	Angle between the vectors	rad

Locals : p,q,w,d,n,s,bMisc Vectors

References :
BMW pg. 109-116 Diagram pg. 109
Escobal pg. 306-307

$$\overline{P} = \bar{r}_2 \times \bar{r}_3$$

$$\overline{Q} = \bar{r}_3 \times \bar{r}_1$$

$$\overline{W} = \bar{r}_1 \times \bar{r}_2$$

Check that the vectors are Coplanar. $\overline{P}$ is $\perp$ to $\bar{r}_2$ and $\bar{r}_3$, so $\overline{P} \cdot \bar{r}_1$ must equal 0 for the vectors to be Coplanar.

$$\overline{D} = \overline{P} + \overline{Q} + \overline{W}$$

$$\overline{N} = r_1\overline{P} + r_2\overline{Q} + r_3\overline{W}$$

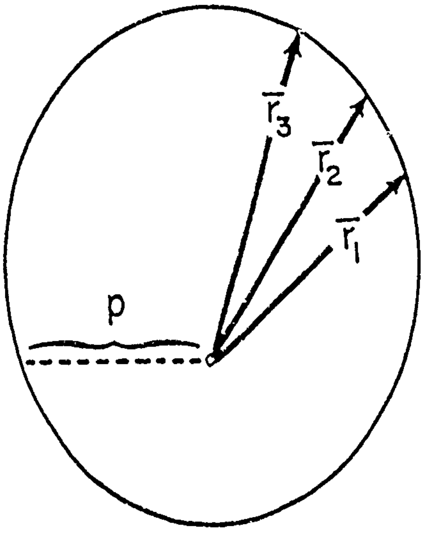
Check that the orbit is possible. $\overline{D}$ and $\overline{N}$ must be non-zero, i.e. the vectors are not Colinear and they must be in the same direction, $\overline{D} \cdot \overline{N} \geq 0.0$.

$$\overline{S} = (r_2 - r_3)\ \overline{P} + (r_3 - r_1)\ \overline{Q} + (r_1 - r_2)\ \overline{W}$$

$$\overline{B} = \overline{D} \times \bar{r}_2$$

$$L = \sqrt{\frac{\mu}{D\,N}}$$

$$\bar{v}_2 = \frac{L}{r_2}\overline{B} + L\,\overline{S}$$



HERRGIBBS(r1,r2,r3,JD1,JD2,JD3,   v2,Theta)

This procedure implements the Herrick-Gibbs approximation for orbit determination, and finds the middle velocity vector for the 3 given position vectors. The method is good for fast calculations and small angles, <= 10 deg. Notice the angle between vectors is passed back to allow the user to make a decision about the accuracy of the results since vectors about 12 deg apart may be accurate, while vectors 170 deg apart would not. The observations MUST be sequential and taken on one revolution. The Use of Julian Dates for input makes it much easier to perform calculations where the sights occur around midnight.

			Variable	Range
Inputs	:R1	- $\bar{r}_1$	IJK Position vector #1	DU
	R2	- $\bar{r}_2$	IJK Position vector #2	DU
	R3	- $\bar{r}_3$	IJK Position vector #3	DU
	JD1	- Julian Date of 1st sighting		days from 4713 B.C.
	JD2	- Julian Date of 2nd sighting		days from 4713 B.C.
	JD3	- Julian Date of 3rd sighting		days from 4713 B.C.
Outputs	:V2	- $\bar{v}_2$	IJK Velocity Vector for R2	DU / TU
	Theta	-	Angle between the vectors	rad
Locals	angl	- Angle between r1 and r2		rad
	ang2	- Angle between r2 and r3		rad
	p,w	- Vectors		

References :
Escobal   pg. 254-256, 304-306

$$\overline{P} = \bar{r}_2 \times \bar{r}_3$$
$$\overline{Q} = \bar{r}_3 \times \bar{r}_1$$

Check that the vectors are Coplanar. $\overline{P}$ is $\perp$ to $\bar{r}_2$ and $\bar{r}_3$, so $\overline{P} \cdot \bar{r}_1$ must equal 0 for the vectors to be Coplanar.

$$\text{Angl} = \left| \text{Cos}^{-1}\left(\frac{P}{\bar{r}_2 \bar{r}_3} \right) \right|$$

$$\text{Ang2} = \left| \text{Cos}^{-1}\left(\frac{P}{\bar{r}_2 \bar{r}_3} \right) \right|$$

Check for the amount of space between the vectors. If the distance is too great, the accuracy could be a problem.

$$\begin{aligned} \bar{v}_2 = \quad & -(\text{t3-t2}) \left(\frac{1}{(\text{t2-t1})(\text{t3-t1})} + \frac{1}{12 r_1^{\,3}} \right) \bar{r}_1 \\ & + ((\text{t3-t2}) - (\text{t2-t1})) \left(\frac{1}{(\text{t2-t1})(\text{t3-t2})} + \frac{1}{12 r_2^{\,3}} \right) \bar{r}_2 \\ & + (\text{t2-t1}) \left(\frac{1}{(\text{t3-t2})(\text{t3-t1})} + \frac{1}{12 r_3^{\,3}} \right) \bar{r}_3 \end{aligned}$$

FINDCandS (Znew, Cnew,Snew)

This procedure calculates the C and S functions for use in the Universal Variable calculations. NOTE equality is handled by the series expansion terms to eliminate potential discontinuities. The series is only used for negative values of Z since the truncation results in rather large errors as Z gets larger than about 10.0.

			Variable
Inputs	:ZNew	-	Z variable
Outputs	:CNew	-	C function value
	SNew	-	S function value

References :

BMW	pg. 207-210	Diagram pg. 209
Kaplan	pg. 304-305	

If $Z_n \leq 0.0$

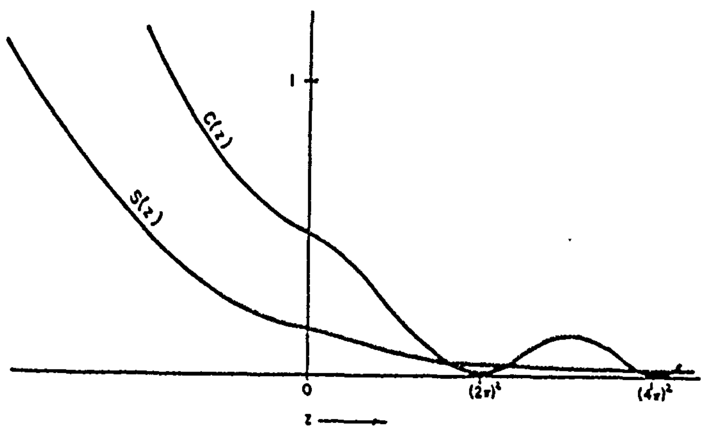
$$C = \frac{1}{2!} - \frac{Z_n}{4!} + \frac{Z_n^{\,2}}{6!} - \frac{Z_n^{\,3}}{8!} + \frac{Z_n^{\,4}}{10!} - \frac{Z_n^{\,5}}{12!} + \cdots$$

$$S = \frac{1}{3!} - \frac{Z_n}{5!} + \frac{Z_n^{\,2}}{7!} - \frac{Z_n^{\,3}}{9!} + \frac{Z_n^{\,4}}{11!} - \frac{Z_n^{\,5}}{13!} + \cdots$$

If $Z_n > 0.0$

$$C = \frac{1 - \mathrm{Cos}(\sqrt{Z_n})}{Z_n}$$

$$S = \frac{\sqrt{Z_n} - \mathrm{Sin}(\sqrt{Z_n})}{\sqrt{Z_n^{\,3}}}$$



Keplers Equation - NEWTONR (e,M, E0,Nu)

This procedure performs the Newton Rhapson iteration to find the Eccentric Anomaly given the Mean anomaly. The True Anomaly is also calculated.

			Variable	Range
Inputs	:e	-	Eccentricity	0.0 - 1.0
	M	-	Mean Anomaly	0.0 - 2Pi rad
Outputs	:E0	- E_o	Eccentric Anomaly	0.0 - 2Pi rad
	Nu	- ν	True Anomaly	0.0 - 2Pi rad
Locals	E1	- E_1	Eccentric Anomaly, next value	rad

References :

BMW pg. 184-187, 220-222 Diagram pg. 221

$E_1 = M$

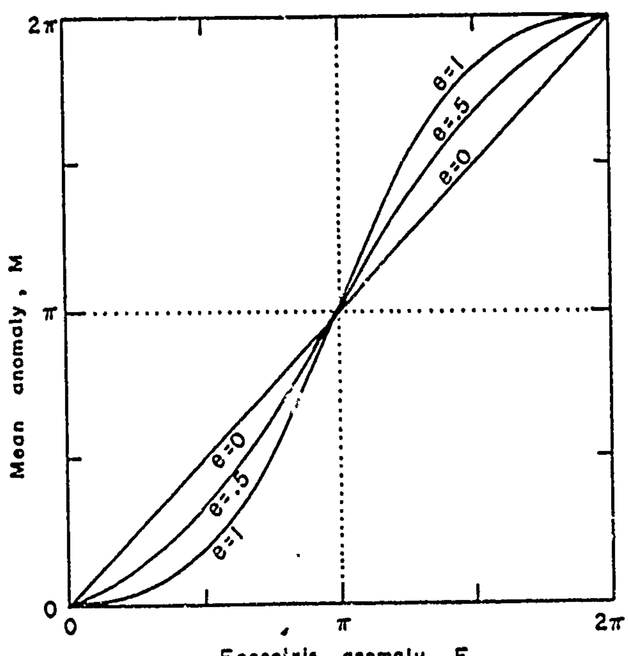
LOOP

$$E_o = E_1$$

$$E_1 = E_o - \frac{E_o - e \operatorname{Sin}(E_o) - M}{1 - e \operatorname{Cos}(E_o)}$$

UNTIL $| E_1 - E_o | < 0.0000001$

$$\nu = \operatorname{Atan2}\left(\frac{\sqrt{1-e^2} \operatorname{Sin}(E_1)}{1 - e \operatorname{Cos}(E_1)} , \frac{\operatorname{Cos}(E_1) - e}{1 - e \operatorname{Cos}(E_1)} \right)$$



KEPLER (ro,vo,t, r,v)

This procedure solves Keplers problem for orbit determination and returns a future Geocentric Equatorial (IJK) position and velocity vector. The solution procedure uses Universal variables.

			Variable	Range
Inputs	:Ro	- $\bar{r}_0$	IJK Position vector - initial	DU
	Vo	- $\bar{v}_0$	IJK Velocity vector - initial	DU / TU
	t	- t	Length of time to propagate	TU
Outputs	:R	- $\bar{r}$	IJK Position vector	DU
	V	- $\bar{v}$	IJK Velocity vector	DU / TU
Locals	F,G	-	f and g expressions	
	FDot,GDot	- $\dot{F},\dot{G}$	Derivatives of f and g expressions	
	XOld	- x_0	Old Universal Variable X	
	XNew	- x_n	New Universal Variable X	
	ZNew	- Z_n	New value of z	
	CNew	- C	C(z) function	
	SNew	- S	S(z) function	
	DeltaT	- dt	change in t	TU
	TimeNew	- t_n	New time	TU
	A	-	Semi major axis	DU
	Alpha	- α	Reciprocol 1/a	
	SME	- $\mathcal{E}$	Specific Mechanical Energy	DU^2 / TU^2
	S	-	Variable for parabolic case	
	W	-	Variable for parabolic case	

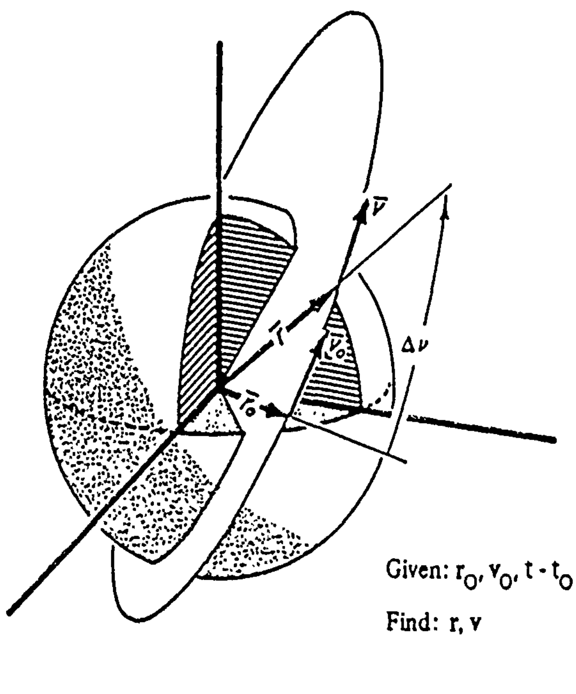
Coupling :

FindCandS Find C and S functions

References :

Kaplan pg. 304-308 (Includes first guess for x if parabolic)

BMW pg. 191-199, 203-212 Diagram pg. 195



Given: $r_0, v_0, t - t_0$

Find: r, v

$$\mathcal{E} = \frac{v_0^2}{2} - \frac{\mu}{r_0}$$

$$a = -\frac{\mu}{2\mathcal{E}} \qquad \alpha = \frac{1}{a}$$

Set up first guess as follows : NOTE since t_0 is 0.0, $t-t_0$ reduces to t

Circle or Ellipse :

$$x_0 \approx \sqrt{\mu}\,(t-t_0)\,\alpha$$

Check if $\alpha = 1.0$ since this makes the first guess too close to converge.

Parabola :

$$\mathrm{Cot}(2s) = 3\sqrt{\frac{\mu}{p^3}}\,(t-t_0)$$

$$\mathrm{Tan}^3 w = \mathrm{Tan}\, s$$

$$\mathrm{Tan}(\tfrac{\nu}{2}) = 2\,\mathrm{Cot}(2w)$$

$$x_0 \approx \sqrt{p}\left[\mathrm{Tan}(\tfrac{\nu}{2}) - \mathrm{Tan}(\tfrac{\nu_0}{2})\right]$$

Hyperbola :

$$x_0 \approx \mathrm{sign}(t-t_0)\sqrt{\frac{-1}{\alpha}}\,\ln\left[\frac{-2\mu\alpha\,(t-t_0)}{\bar{r}_0\cdot\bar{v}_0 + \mathrm{sign}(t-t_0)\sqrt{\frac{-\mu}{\alpha}}\,(1 - r_0\alpha)}\right]$$

LOOP

$$Z_n = x_0^2\,\alpha$$

If $Z_n \le 0.0$

$$C = \frac{1}{2!} - \frac{Z_n}{4!} + \frac{Z_n^2}{6!} - \frac{Z_n^3}{8!} + \cdots \qquad S = \frac{1}{3!} - \frac{Z_n}{5!} + \frac{Z_n^2}{7!} - \frac{Z_n^3}{9!} + \cdots$$

If $Z_n > 0.0$

$$C = \frac{1 - \mathrm{Cos}(\sqrt{Z_n})}{Z_n} \qquad S = \frac{\sqrt{Z_n} - \mathrm{Sin}(\sqrt{Z_n})}{\sqrt{Z_n^3}}$$

$$t_n = \frac{x_0^3 S + \frac{\bar{r}_0\cdot\bar{v}_0}{\sqrt{\mu}}\,x_0^2\,C + r_0 x_0(1 - Z_n S)}{\sqrt{\mu}}$$

$$dt = \frac{x_0^2 C + \frac{\bar{r}_0\cdot\bar{v}_0}{\sqrt{\mu}}\,x_0\,(1 - Z_n S) + r_0(1 - Z_n C)}{\sqrt{\mu}}$$

$$x_n = x_0 + \frac{t-t_n}{dt}$$

Check if elliptical orbit ($A > 0.0$ and $\mathcal{E} < 0.0$) and $x_n > 2\pi\sqrt{a}$. If so, change dt so the iteration doesn't converge as quickly. A value of (10.0)dt in the preceding equation seems to work.

$$x_0 = x_n$$

UNTIL $|t - t_n| < 0.00001$

$$f = 1 - \frac{x_0^2}{r_0}\,C \qquad g = t - \frac{x_0^3}{\sqrt{\mu}}\,S$$

$$\boxed{\bar{r} = f\,\bar{r}_0 + g\,\bar{v}_0}$$

$$\dot{g} = 1 - \frac{x^2}{r}\,C \qquad \dot{f} = \frac{\sqrt{\mu}}{r\,r_0}\,x_0\,(Z_n S - 1)$$

$$\boxed{\bar{v} = \dot{f}\,\bar{r}_0 + \dot{g}\,\bar{v}_0}$$

GAUSS (r1,r2,dm,time, v1,v2)

This procedure solves the Gauss problem of orbit determination and returns the velocity vectors at each of two given position vectors. The solution uses Universal Variables for calculation and a bisection technique for updating Z. This method is slower than the Newton iteration discussed in BMW, but it does NOT suffer problems with negative z values, and is valid for ellipses LESS THAN one revolution, parabolas, and Hyperbolas. Also note the selection of small since the algorithm is very sensitive to changes in this variable. A value of 0.001 will converge in say 10 iterations instead of 25 iterations with a value of 0.00001, and the accuracy will differ in the 3rd-4th decimal place. I chose to keep the higher accuracy for cases like the example, BMW pg. 274, #5.10.

(Refer to graph on BMW pg. 235 for ranges of z.)

			Variable	Range
Inputs :	R1	- $\bar{r}_{int}$	IJK Position vector of interceptor	DU
	R2	- $\bar{r}_{tgt1}$	IJK Position vector of target after time	DU
	DM	-	direction of motion	'L','S'
	Time	- t_0	Time between R1 and R2	TU
OutPuts :	V1	- $\bar{v}_{1tran}$	IJK Velocity vector of transfer orbit	DU / TU
	V2	- $\bar{v}_{2tran}$	IJK Velocity vector of transfer orbit	DU / TU
Local Variables :				
	VarA	-	Variable of the iteration, NOT the semi major axis!	
	Y	- y		
	F,G	- f,g	f and g expressions	
	GDot	- $\dot{g}$	Derivative of g expression	
	XOld	- x_0	Universal Variable X	
	ZOld	- Z_0	New value of z	
	ZNew	- Z_n	New value of z	
	CNew	- C	C(z) function	
	SNew	- S	S(z) function	
	TimeNew	- t	New time	TU

References :

BMW pg. 228-241 (Uses a Newton iteration) Diagram pg. 235

GAUSS ($\bar{r}_1, \bar{r}_2$, dm, t_o, $\bar{v}_1, \bar{v}_2$)

$\text{Cos } \Delta\nu = \frac{\bar{r}_1 \cdot \bar{r}_2}{r_1 r_2}$

dm = + (Short Way) or – (Long Way)

$\text{VarA} = \text{dm}\sqrt{r_1 r_2 (1 + \text{Cos}(\Delta\nu))}$ If VarA = 0.0, the orbit is not possible

Guess $Z_0 = 0.0$, therefore $C = \frac{1}{2}$ and $S = \frac{1}{6}$

Set bounds: Upper $= 4\pi^2$ and Lower $= -4\pi$

LOOP:

$$y_n = r_1 + r_2 - \frac{\text{VarA}(1 \quad Z_oS)}{\sqrt{C}}$$

Check if VarA > 0.0 and y < 0.0, then re-adjust lower bound of Z until y > 0.0

$$x_o = \sqrt{\frac{y_n}{C}}$$

$$t = \frac{x_o^3 S + \text{VarA}\sqrt{y_n}}{\sqrt{\mu}}$$

IF $t < t_0$ reset lower bound $= Z_0$
else
IF $t > t_0$ reset upper bound $= Z_0$

$$Z_n = \frac{\text{upper} + \text{lower}}{2}$$

Calculate C and S:

If $Z_n \leq 0.0$

$$C = \frac{1}{2!} - \frac{Z_n}{4!} + \frac{Z_n^2}{6!} - \frac{Z_n^3}{8!} + \cdots \qquad S = \frac{1}{3!} - \frac{Z_n}{5!} + \frac{Z_n^2}{7!} - \frac{Z_n^3}{9!} + \cdots$$

If $Z_n > 0.0$

$$C = \frac{1 - \text{Cos}(\sqrt{Z_n})}{Z_n} \qquad S = \frac{\sqrt{Z_n} - \text{Sin}(\sqrt{Z_n})}{\sqrt{Z_n^3}}$$

$$Z_o = Z_n$$

Check if the first guess is too close

UNTIL $| t - t_o | < 0.00001$

Evaluate f and g coefficients

$$f = 1 - \frac{y_n}{r_1} \qquad g = \text{VarA}\sqrt{\frac{y_n}{\mu}}$$

$$\boxed{\bar{v}_1 = \frac{\bar{r}_2 - f\,\bar{r}_1}{g}}$$

$$\dot{g} = 1 - \frac{y_n}{r_2}$$

$$\boxed{\bar{v}_2 = \frac{\dot{g}\,\bar{r}_2 - \bar{r}_1}{g}}$$

IJKtoLATLON (R,JD,   Latgc,Lon)

This procedure converts a Geocentric Equatorial (IJK) position vector into latitude and longitude. Geodetic and Geocentric latitude are found.

			Variable	Range
Inputs	:R	- $\bar{r}$	IJK Position Vector	DU
	JD	-	Julian Date	days from 4713 B.C.
Outputs	:GeoCnLat -		Geocentric Latitude	$-\pi/2$ to $\pi/2$ rad
	Lon	-	Longitude (WEST -)	-2π to 2π rad
Locals	:			
	Rc	-	Range of site w.r.t. earth center	DU
	Height	-	Height above earth w.r.t. site	DU
	Alpha	- α	Angle from I axis to point, LST	rad
	DeltaLat	- Δ Lat	Diff between Delta and Geocentric lat	rad
	GeoDtLat	- Lat_{gd}	Geodetic Latitude	rad
	Delta	- δ	Declination angle of R1 in IJK system	rad
	AE	- a_e	Equatorial radius of Earth	DU
	GST	-	Greenwich Sidereal Time	rad
Constants	:			
	Flat	- f	Flatenning of the Earth	0.003352810664747352
Coupling :				
GSTime		Greenwich Sidereal Time		
References :				
Escobal		pg. 398-399		

$$r = \sqrt{r_i^2 + r_j^2 + r_k^2}$$

$$\alpha = \mathrm{Tan}^{-1}\left(\frac{r_j}{r_i}\right)$$

Use procedure GSTime to Find GST

Long $= \alpha - $ GST

$$\delta = \mathrm{Tan}^{-1}\left(\frac{r_k}{\sqrt{r_i^2 + r_j^2}}\right)$$

Let $Lat_{gc} = \delta$

LOOP

$$R_c = a_e\sqrt{\frac{1-(2f-f^2)}{1-(2f-f^2)\,\mathrm{Cos}^2(Lat_{gc})}}$$

$$Lat_{gd} = \mathrm{Tan}^{-1}\left(\frac{1}{(1-f)^2}\mathrm{Tan}(Lat_{gc})\right)$$

$$\mathrm{Height} = \sqrt{r^2 - R_c^2\mathrm{Sin}^2(Lat_{gd} - Lat_{gc})} - R_c\mathrm{Cos}(Lat_{gd} - Lat_{gc})$$

$$\Delta Lat = \mathrm{Sin}^{-1}\left(\frac{\mathrm{Height}}{r}\,\mathrm{Sin}\,(Lat_{gd} - Lat_{gc})\right)$$

$$Lat_{gc} = \delta - \Delta Lat$$

UNTIL $\Delta Lat_{n-1} - \Delta Lat_n < 0.00001$

RNGAZ (Llat,Llon,Tlat,Tlon,TOF, Range,Az)

This procedure calculates the Range and Azimuth between two specified ground points. Notice the range will ALWAYS be within the range of values listed since you do not know the direction of firing, long or short. The procedure will calculate Rotating Earth ranges if the TOF is passed in.

Inputs :

LLat	-	Start Geocentric Latitude	$-\pi/2$ - $\pi/2$ rad
LLon	-	Start Longitude (WEST -)	0.0 - 2π rad
TLat	-	End Geocentric Latitude	$-\pi/2$ - $\pi/2$ rad
TLon	-	End Longitude (WEST -)	0.0 - 2π rad
TOF	-	Time of Flight if ICBM, 0.0 otherwise	TU

Outputs :

Range	- Λ	Range between points	0.0 - π rad
Az	- β	Azimuth	0.0 - 2π rad

Constants :

OmegaEarthω_0	Angluar Rotation of the Earth	0.058833590688786 rad/TU

References :
BMW pg. 309-311

$$\Lambda = \operatorname{Cos}^{-1}\Big(\operatorname{Sin}(\text{Llat})\operatorname{Sin}(\text{TLat}) + \operatorname{Cos}(\text{Llat})\operatorname{Cos}(\text{Tlat})\operatorname{Cos}(\text{Tlon}-\text{Llon}+\omega_0\text{TOF})\Big)$$

Check for singular values of Range, 0.0 or half the distance around the Earth

$$\beta = \operatorname{Cos}^{-1}\left(\frac{\operatorname{Sin}(\text{Tlat}) - \operatorname{Cos}(\Lambda)\operatorname{Sin}(\text{Tlat})}{\operatorname{Sin}(\Lambda)\operatorname{Cos}(\text{Llat})}\right)$$

Check if the Azimuth is greater than 180 degrees by

IF Sin(Tlon -Llon + ω_0TOF) < 0.0 THEN

$$\beta = 2\pi - \beta$$

PATH (Llat,Llon,Range,Az, Tlat,Tlon)

This procedure determines the end position for a given range and azimuth from a given point. Notice the use of ATAN2 to eliminate quadrant problems. Also, Geocentric coordinates are used since the Earth is assumed to be spherical.

Inputs :

LLat	-	Start Geocentric Latitude	$-\pi/2$ - $\pi/2$ rad
LLon	-	Start Longitude	0.0 - 2π rad
Range	- Λ	Range between points	DU
Az	- β	Azimuth	0.0 - 2π rad

Outputs :

TLat	-	End Geocentric Latitude	$-\pi/2$ - $\pi/2$ rad
TLon	-	End Longitude	0.0 - 2π rad

Locals :

DeltaN	- ΔN	Angle bteween the two points	rad

Coupling :

Atan2 — Arc Tangent function which also resolves quadrants

References :

BMW pg. 309-311

Make sure Azimuth, Llon, and Range are within first quadrant constraints

$$\text{Tlat} = \text{Sin}^{-1}\Big(\text{Sin}(\text{ Llat }) \text{ Cos}(\ \Lambda\ ) + \text{Cos}(\text{ Llat }) \text{ Sin}(\ \Lambda\ ) \text{ Cos}(\ \beta\ ) \Big)$$

$$\Delta \text{N} = \text{ATAN2}\left(\frac{\text{Sin}(\text{ Az }) \text{ Sin}(\ \Lambda\ )}{\text{Cos}(\text{ Tlat })}, \frac{\text{Cos}(\ \Lambda\ ) - \text{Sin}(\text{ Tlat }) \text{ Sin}(\text{ Llat })}{\text{Cos}(\text{ Tlat }) \text{ Cos}(\text{ Llat })} \right)$$

$$\text{Tlon} = \text{Llon} + \Delta \text{N}$$

Check quadrants of Tlon

TRAJEC (Llat,Llon,Tlat,Tlon,Rbo,Q,TypePhi, Range,Phi,TOF,Az,ICPhi,ICVbo,ICRbo,Vn)

This procedure calculates the Range, Azimuth, and Time of Flight between two specified ground points for an ICBM with as known Q. Calculations depend on knowledge of burnout conditions, and the iterations are performed for either a high or low trajectory. Notice the ICBM will fly on an inertial trajectory, and values for earth relative velocities, etc., are calculated after the iteration. Notice these calculations do not support trajectories over half the world away.

Inputs	:LLat	-	Start Geocentric Latitude	$-\pi/2$ - $\pi/2$ rad
	LLon	-	Start Longitude (WEST -)	0.0 - 2π rad
	TLat	-	End Geocentric Latitude	$-\pi/2$ - π rad
	TLon	-	End Longitude (WEST -)	0.0 - 2π rad
	Rbo	-	Radius at burnout	DU
	Q	-	Non-dimensional Q performance based on Inertial Velocity	
	TypePhi	-	Type of trajectory, High ot Low	'H', 'L'
Outputs	:Range	- Λ	Rotating Range between points	0.0 - π rad
	Phi	- ϕ	Inert Flight Path Angle	rad
	TOF	-	Rotating Earth Time of Fligth	TU
	Az	- β	Inert Azimuth	0.0 - 2π rad
	ICPhi	-	Influence Coefficient for Phi	rad/rad
	ICVbo	-	Influence Coefficient for Vbo	rad/ DU/TU
	ICRbo	-	Influence Coefficient for Rbo	rad/rad
	Vn	-	Velocity the missile needs to provide	DU/TU
Locals	:QBoMin	-	Minimum Q for a given range	
	a	-	Semi Major Axis	DU
	Ecc	- e	Eccentricity	
	E	- E	Eccentric Anomaly	rad
	RangeOld	-	Iteration value of range	DU
	Vbo	-	Inertial Velocity	DU/TU
	VEarth	-	Earths velocity	DU/TU

Constants :

OmegaEarth ω_0	Angluar Rotation of the Earth	0.0588335900688786 rad/TU

Coupling :

RngAz — Finds range and Azimuth given two points

References :

BMW pg. 293-313

Find Range and Azimuth using RNGAZ Λ, β

$$a = \frac{r_{bo}}{2 - Q}$$

$$Q_{bo}\min = \frac{2\text{Sin}(\frac{\Lambda}{2})}{1 + \text{Sin}(\frac{\Lambda}{2})}$$

IF The ICBM trajectory is possible ($Q_{bo} > Q_{bo}\min$)

LOOP

IF High Trajectory:

$$\phi = 0.5\left(\pi - \text{Sin}^{-1}(\frac{2-Q}{Q})\ \text{Sin}(\frac{\Lambda}{2}) - \frac{\Lambda}{2}\right)$$

IF Low Trajectory

$$\phi = 0.5\left(\text{Sin}^{-1}(\frac{2-Q}{Q})\ \text{Sin}(\frac{\Lambda}{2}) - \frac{\Lambda}{2}\right)$$

$$e = \sqrt{1 + Q\ (\ Q\text{-}2\ )\ \text{Cos}^2(\ \phi\ )}$$

$$E = \text{Cos}^{-1}\left(\frac{e - \text{Cos}(\frac{\Lambda}{2})}{1 - e\text{Cos}(\frac{\Lambda}{2})}\right)$$

$$TOF = 2\sqrt{\frac{a^3}{\mu}}\ (\ \pi - E + e\ \text{Sin}(\ E\ )\ )$$

Find Range and Azimuth using new TOF Λ, β

UNTIL (RangeOld - Λ) > Small

$$V_{bo} = \sqrt{\frac{Q}{r_{bo}}}$$

Evaluate Influence Coefficients

$$ICv = \frac{8\mu}{v^3_{bo} r_{bo}}\ \frac{\text{Sin}^2(\frac{\Lambda}{2})}{\text{Sin}(\ 2\phi\ )}$$

$$ICr = \frac{4\mu}{v^2_{bo} r^2_{bo}}\ \frac{\text{Sin}^2(\frac{\Lambda}{2})}{\text{Sin}(\ 2\phi\ )}$$

$$IC\phi = \frac{2\text{Sin}(\ \Lambda + 2\phi\ )}{\text{Sin}(\ 2\phi\ )} - 2$$

$$V_{earth} = \omega_o \text{Cos}(\ \text{Llat}\ )$$

$$V_n = \begin{bmatrix} -V_{bo}\text{Cos}(\ \phi\ )\ \text{Cos}(\ \beta\ ) \\ V_{bo}\text{Cos}(\ \phi\ )\ \text{Sin}(\ \beta\ ) \\ V_{bo}\text{Sin}(\ \phi\ ) \end{bmatrix}$$

J2DRAGPERT (Inc,E,N,NDot, OmegaDot,ArgpDot,EDot)

This procedure calculates the perturbations for the predict problem involving secular rates of change resulting from J2 and Drag only.

Inputs	:Inc	- i	Inclination	rad
	e	-	Eccentricity	
	N	-	Mean Motion	rad/TU
	NDot	- $\dot{n}$	Mean Motion rate	rad / 2TU^2
Outputs	:OmegaDot	-	Long of Asc Node rate	rad / TU
	ArgpDot	-	Argument of perigee rate	rad / TU
	EDot	-	Eccentricity rate	/ TU
Locals	p	-	Semi-parameter	DU
	a	-	Semi-major axis	DU
Constants :				
	J2	- J_2	J_2 harmonic of the Earth	0.00108263

References :

Escobal pg. 369

O'Keefe et al., Astronomical J, Vol 64 num 7, pg. 247 for Edot

---

J_2 - First order equations where n = 2

$$a = a_o$$
$$e = e_o$$
$$i = i_o$$
$$\bar{n} = n_o\left[1 + \frac{3}{2} J_2 \frac{\sqrt{1-e^2}}{p^2}\left(1 - \frac{3}{2}\operatorname{Sin}^2 i\right)\right]$$

$$\Omega = \Omega_o - \left(\frac{3}{2}\frac{J_2}{p^2}\operatorname{Cos} i\right)\bar{n}(t - t_o) \qquad \boxed{\dot{\Omega} = -\left(\frac{3}{2}\frac{J_2}{p^2}\operatorname{Cos} i\right)\bar{n}}$$

$$\omega = \omega_o + \left(\frac{3}{2}\frac{J_2}{p^2}\left[2 - \frac{5}{2}\operatorname{Sin}^2 i\right]\right)\bar{n}(t - t_o) \qquad \boxed{\dot{\omega} = \left(\frac{3}{2}\frac{J_2}{p^2}\left[2 - \frac{5}{2}\operatorname{Sin}^2 i\right]\right)\bar{n}}$$

$$M = M_o + \bar{n}(t - t_o)$$

Drag - Simplified by assun ..g radius of perigee is constant as drag reduces semi major axis a, therfore, proceed as:

$$r_p = a(1 - e)$$

$$\Delta r_p = \Delta a(1 - e) - a\Delta e \approx 0$$

$$\cdots$$

$$e = e_o - \frac{2(1 - e)\dot{n}_o}{3\bar{n}}(t - t_o) \qquad \boxed{\dot{e} = -\frac{2(1 - e)\dot{n}_o}{3\bar{n}}}$$

PREDICT(JD,JDEpoch,No,NDot,Eo,EDot,Inco,Omegao, OmegaDot,Argpo,ArgpDot,Mo,Lat,Lon,Alt, Rho,Az,El,Vis)

This procedure determines the azimuth and elevation for the viewing of a staellite from a known ground site. Notice the Julian Date is left in it's usual DAYS format since the dot terms are input as radians per day, thus no extra need for conversion. The Julian Date also facilitates finding the site position vector. Also observe RANDV is not used since this would merely accomplish extra calculations. The iteration is left out to allow the user to set up his own loop to look for possible sighting times.

Inputs	:JD	-	Julian Date of desired observation	Days
	JDEpoch	-	Julian date of epoch for satellite	Days
	No	- n_o	Epoch Mean motion	rad/day
	Ndot	- $\dot{n}_o$	Epcoh Half Mean Motion Rate	rad/2day^2
	Eo	- e_o	Epoch Eccentricity	
	Edot	- $\dot{e}_o$	Epoch Eccentricity rate	/day
	Inco	- i_o	Epoch Inclination	rad
	Omegao	- Ω_o	Epoch Lon of Asc node	rad
	OmegaDot	- $\dot{\Omega}_o$	Epoch Lon of Asc Node rate	rad/day
	Argpo	- ω_o	Epoch Argument of perigee	rad
	ArgpDot	- $\dot{\omega}_o$	Epoch Argument of perigee rate	rad/day
	Mo	- Mo	Epoch Mean Anomaly	rad
	Lat	-	Geodetic Latitude of site	rad
	Lon	-	Longitude of site	rad
	Alt	-	Altitude of site	DU
Outputs	:Rho	- ρ	Range from site to satellite	DU
	Az	- β	Azimuth	rad
	El	-	Elevation	rad
	Vis	-	Visibility	Radar Sun, Eye, Radar Nite, Not Visible
Locals	:Variable o	-	denotes the epoch value, while no o is current	
	Dt	-	Change in time from Epoch to desired t	days
	A	-	Semi major axis	DU
	E0	-	Eccentric Anomaly	rad
	Nu	- ν	True Anomaly	rad
	LST	-	Local Sidereal Time	rad
	GST	-	Greenwich Sidereal Time	rad
	Theta	-	Angle between IJK Sun and Satellite vec	rad
	Dist	-	Ppdculr distance of satellite from RSun	DU
	R	-	IJK Satellite vector	DU
	RS	-	IJK Site Vector	DU
	VS	-	Site Velocity vector	DU/TU
	RhoVec	-	Site to satellite vector in SEZ	DU
	RHoV	-	Site to satellite vector in IJK	DU
	RSun	-	Sun vector	AU

Coupling :

SUN	Position vector of Sun
SITE	Site Vector
LSTime	Local Sidereal Time
NewtonR	Iterate to find Eccentric Anomaly

References :

Escobal pg. 369

$Dt = JD - JDepoch$

Update orbital elements from epoch to current time, JD

$$e = e_o + \dot{e}_o \Delta t$$
$$i = i_o$$
$$\Omega = \Omega_o + \dot{\Omega}_o \Delta t$$
$$\omega = \omega_o + \dot{\omega}_o \Delta t$$
$$n = n_o + \dot{n}_o \Delta t$$
$$M = M_o + n_o \Delta t + \frac{\dot{n}_o}{2} \Delta t^2$$

NOTICE!! The NORAD 2-card element set has $\dot{n}_o$ sent as $\frac{\dot{n}_o}{2}$

Use NEWTONR to Find Eccentric Anomaly and True Anomaly E and ν, Then find position vector

$$a = n^{-2/3}$$

$$\bar{r}_{pqw} = \begin{bmatrix} \dfrac{p\, \mathrm{Cos}(\nu)}{1 + e\mathrm{Cos}(\nu)} \\ \dfrac{p\, \mathrm{Sin}(\nu)}{1 + e\mathrm{Cos}(\nu)} \\ 0 \end{bmatrix}$$

$$\bar{R}_{ijk} = \begin{bmatrix} \mathrm{Cos}\,\Omega \mathrm{Cos}\,\omega - \mathrm{Sin}\,\Omega \mathrm{Sin}\,\omega \mathrm{Cos}\,i & -\mathrm{Cos}\,\Omega \mathrm{Sin}\,\omega - \mathrm{Sin}\,\Omega \mathrm{Cos}\,\omega \mathrm{Cos}\,i & +\mathrm{Sin}\,\Omega \mathrm{Sin}\,i \\ \mathrm{Sin}\,\Omega \mathrm{Cos}\,\omega + \mathrm{Cos}\,\Omega \mathrm{Sin}\,\omega \mathrm{Cos}\,i & -\mathrm{Sin}\,\Omega \mathrm{Sin}\,\omega + \mathrm{Cos}\,\Omega \mathrm{Cos}\,\omega \mathrm{Cos}\,i & -\mathrm{Cos}\,\Omega \mathrm{Sin}\,i \\ \mathrm{Sin}\,\omega \mathrm{Sin}\,i & \mathrm{Cos}\,\omega \mathrm{Sin}\,i & \mathrm{Cos}\,i \end{bmatrix} \bar{R}_{pqw}$$

Use LSTIME and SITE to find the LST and the SIte Vector $\overline{RS}_{ijk}$

$$\bar{\rho}_{ijk} = \bar{r}_{ijk} - \overline{RS}_{ijk}$$

Find Topocentric Right ascension and Declination

$$RtAsc = ATan2\left(\frac{\rho_j}{\sqrt{\rho_i^2 + \rho_j^2}}, \frac{\rho_i}{\sqrt{\rho_i^2 + \rho_j^2}\,\rho} \right)$$

$$Decl = \mathrm{Sin}^{-1}\left(\frac{\rho_k}{\rho} \right)$$

$$\bar{\rho}_{sez} = \begin{bmatrix} \mathrm{Sin(lat)Cos(lst)} & \mathrm{Sin(lat)Sin(lst)} & \mathrm{-Cos(lat)} \\ \mathrm{-Sin(lst)} & \mathrm{Cos(lst)} & 0 \\ \mathrm{Cos(lat)Cos(lst)} & \mathrm{Cos(lat)Sin(lst)} & \mathrm{Sin(lat)} \end{bmatrix} \bar{\rho}_{ijk}$$

Check Visibility

IF Above the Horizon

- $\bar{\rho}_{sez}[z] > 0.0$
- IF Night Time at the site
  - $\overline{RS}_{ijk} \cdot \overline{RSun}_{ijk} < 0.0$
  - IF Satellite not in Earth's shadow
    - Find angle θ between $\bar{R}$ and $\overline{RSun}$
    - Dist = R Cos($\theta - 90^\circ$)
    - If Dist > 1.0 (DU's), satellite is visible to the eye

Find Topocentric Azimuth and Elevation

$$Az = ATan2\left(\frac{\rho_e}{\sqrt{\rho_s^2 + \rho_e^2}}, \frac{-\rho_s}{\sqrt{\rho_s^2 + \rho_e^2}} \right)$$

$$El = \mathrm{Sin}^{-1}\left(\frac{\rho_z}{\rho} \right)$$

RENDEZVOUS (Rcs1,Rcs2,PhaseI,NumRevs, PhaseF,Waitime)

This procedure calculates the parameters for a Hohmann transfer type rendezvous.

Inputs	:Rcs1	r_1-	Radius of circular interceptor	DU
	Rcs2	r_2-	Radius of circular target	DU
	PhaseI	- ϕ_i	Initial Phase angle	rad
	NumRevs	-	Number of revs to wait	
Outputs	:PhaseF	- ϕ_f	Final Phase Angle	rad
	WaitTime	-	Wait time until next opportunity	TU
Locals	:LeadAng	- α	Lead Angle	rad
	A	- a_2	Semi-major axis	DU

Constants :

References :

For the Transfer orbit :

$$a_2 = \frac{r_1 + r_2}{2}$$

Time of flight for a Hohmann transfer is half the period of the transfer orbit :

$$\text{TOF} = \pi\sqrt{\frac{a_2^3}{\mu}}$$

From the formula for circular satellite speed :

$$v_{tgt} = \sqrt{\frac{\mu}{r_1}}$$

$$v_{int} = \sqrt{\frac{\mu}{r_2}}$$

$$\alpha = V_{tgt}\,\text{TOF}$$

$$\phi_f = \pi - \alpha$$

$$\text{Wait} = \frac{\phi_i - \phi_f + 2\pi\ \text{NumRevs}}{v_{int} - v_{tgt}}$$

Hohmann (R1,R3,e1,e3,Nu1,Nu3, DeltaV1,DeltaV2,TOF)

This procedure calculates the delta v's for a Hohmann transfer for either circle to cirlce, or ellipse to ellipse. The notation used is from the initial orbit (1) at point a, trasnfer is made to the trasnfer orbit (2), and to the final orbit (3) at point b.

Inputs	:R1	-	Initial position magnitude	DU
	R3	-	Initial position magnitude	DU
	e1	-	Eccentricity of orbit 1	
	e3	-	Eccentricity of orbit 3	
	Nu1	-	True Anomaly of orbit 1	rad
	Nu3	-	True Anomaly of orbit 3	rad
Outputs	:DeltaVa	-	Change of velocity at a	DU / TU
	DeltaVb	-	Change of velocity at b	DU / TU
	TOF	-	Time of transfer	TU
Locals	:V1a	- v_{1a}	velocity at a	DU / TU
	V2a	- v_{2a}	velocity at a	DU / TU
	V2b	- v_{2b}	velocity at b	DU / TU
	V3b	- v_{3b}	velocity at b	DU / TU
	A	- a_2	Semi-major axis	DU
References :				

From the formula for circular satellite speed :

$$v_{1a} = \sqrt{\frac{\mu}{r_1}}$$

$$v_{3b} = \sqrt{\frac{\mu}{r_3}}$$

For the Transfer orbit :

$$a_2 = \frac{r_1 + r_3}{2}$$

From the equation for elliptical satellite speed :

$$v_{2a} = \sqrt{2\frac{\mu}{r_1} - \frac{\mu}{a_2}}$$

$$v_{2b} = \sqrt{2\frac{\mu}{r_3} - \frac{\mu}{a_2}}$$

$$\Delta v = |\ v_{2a} - v_{1a}\ | + |\ v_{3b} - v_{2b}\ |$$

Time of flight for a Hohmann transfer is half the period of the transfer orbit :

$$\frac{P}{2} = \pi\sqrt{\frac{a_2{}^3}{\mu}}$$

ONE TANGENT (R1,R3,e1,e3,Nu1,Nu2,Nu3, DeltaV1,DeltaV2,TOF)

This procedure calculates the delta v's for a one tangent transfer for either circle to cirlce, or ellipse to ellipse. The notation used is from the initial orbit (1) at point a, trasnfer is made to the trasnfer orbit (2), and to the final orbit (3) at point b.

Inputs	:R1	-	Initial position magnitude	DU
	R3	-	Initial position magnitude	DU
	e1	-	Eccentricity of orbit 1	
	e3	-	Eccentricity of orbit 3	
	Nu1	-	True Anomaly of orbit 1	rad
	Nu2	-	True Anomaly of orbit 2	rad
	Nu3	-	True Anomaly of orbit 3	rad
Outputs	:DeltaVa	-	Change of velocity at a	DU / TU
	DeltaVb	-	Change of velocity at b	DU / TU
	TOF	-	Time of transfer	TU
Locals	:V1a	- v_{1a}	velocity at a	DU / TU
	V2a	- v_{2a}	velocity at a	DU / TU
	V2b	- v_{2b}	velocity at b	DU / TU
	V3b	- v_{3b}	velocity at b	DU / TU
	A	- a_2	Semi-major axis	DU
References :				

Consider the one tangent burn transfer illustrated to the right. Before determining the total change in velocity, the transfer orbit eccentricity must be calculated.

$$e_2 = \frac{\frac{r_1}{r_3} - 1}{\text{Cos}\,\nu_{2b} - \frac{r_1}{r_3}} \qquad r_1 = a_2\,(1 - e_2) \qquad a_2 = \frac{r_1}{(1 - e_2)}$$

From the formula for circular satellite speed :

$$v_{1a} = \sqrt{\frac{\mu}{r_1}} \qquad v_{3b} = \sqrt{\frac{\mu}{r_3}}$$

From the equation for elliptical satellite speed :

$$v_{2a} = \sqrt{2\frac{\mu}{r_1} - \frac{\mu}{a_2}} \qquad v_{2b} = \sqrt{2\frac{\mu}{r_3} - \frac{\mu}{a_2}}$$

$$\Delta v_a = |\ v_{2a} - v_{1a}\ |$$

The flight path angle is needed for the non-tangential transfer at b

$$\text{Tan}\,\phi_{2b} = \frac{e_2\,\text{Sin}\,\nu_{2b}}{1 + e_2\,\text{Cos}\,\nu_{2b}}$$

Since the final orbit is circular, $\phi_{3b} = 0.0$.

$$\Delta v_b = \sqrt{v_{3b}^2 + v_{2b}^2 - 2v_{3b}v_{2b}\text{Cos}\,(\phi_{3b} - \phi_{2b})}$$

The total Δ v is simply the sum of the two burns.

$$\Delta v = \Delta v_a + \Delta v_b$$

Time of flight is calculated using Keplers Equation.

$$TOF = \sqrt{\frac{a_2^3}{\mu}}\left(2k\pi + (E - e_2\,\text{Sin}\,E) - (E_o - e_2\,\text{Sin}\,E_o)\right)$$

Since this transfer is initiated at periapsis, $E_o = 0.0$. The transfer does not pass periapsis, so k must = zero.

$$\text{Cos}\,E = \frac{e_2 + \text{Cos}\,\nu_{2b}}{1 + e_2\,\text{Cos}\,\nu_{2b}}$$

HILLSR (R,V,Alt,T, R1,V1)

This procedure calculates the various position information for Hills equations. Notice the XYZ system used has Y colinear with Target Position vector, Z normal to target orbit plane, and X in the direction of velocity.

Inputs	:R	-	Initial position vector of INT	DU
	V	-	Initial velocity vector of INT	DU/TU
	Alt	-	Altitude of target satellite	DU
	T	-	Desired Time	TU
Outputs	:R1	-	Final position vector of INT	DU
	V1	-	Final velocity vector of INT	DU/TU
Locals	n	-	Circular velocity of INT	DU/TU

References :

Kaplan pg. 108-115

$$n = \sqrt{\frac{1}{r}}$$

$$x(t) = \frac{\dot{x}_0}{n}\,\text{Sin}\ nt - \left(\frac{2\dot{y}_0}{n} + 3x_0\right)\text{Cos}\ nt + \left(\frac{2\dot{y}_0}{n} + 4x_0\right)$$

$$y(t) = \frac{2\dot{x}_0}{n}\,\text{Cos}\ nt + \left(\frac{4\dot{y}_0}{n} + 6x_0\right)\text{Sin}\ nt + \left(y_0 - \frac{2\dot{x}_0}{n}\right) - \left(3\dot{y}_0 + 6nx_0\right)t$$

$$z(t) = z_0\,\text{Cos}\ nt + \frac{\dot{z}_0}{n}\,\text{Sin}\ nt$$

HILLSV (R,Alt,T, V)

This procedure calculates the initial velocity for the Hills equations. Notice the XYZ system used has Y colinear with Target Position vector, Z normal to target orbit plane, and X in the direction of velocity.

Inputs	:R	-	Initial position vector of INT	DU
	Alt	-	Altitude of target satellite	DU
	T	-	Desired Time	TU
Outputs	:V	-	Initial velocity vector of INT	DU/TU
Locals	n	-	Circular velocity of INT	DU/TU

References :

Kaplan pg. 108-115

$$n = \sqrt{\frac{1}{r}}$$

$$\dot{y}_0 = \frac{\left(6x_0\,(nt - \text{Sin}\,nt) - y_0\right) n\text{Sin}\,nt - 2nx_0\,(4 - 3\text{Cos}\,nt)(1 - \text{Cos}\,nt)}{(4\text{Sin}\,nt - 3nt)\text{Sin}\,nt + 4(1 - \text{Cos}\,nt\,)^2}$$

$$\dot{x}_0 = \frac{nx_0\,(4 - 3\text{Cos}\,nt) + 2(1 - \text{Cos}\,nt)\,\dot{y}_0}{\text{Sin}\,nt}$$

REENTRY (Vre,PhiRe,BC,H, V,Decl,MaxDecl)

This procedure calculates various reentry paramters using the Allen & Eggars approximations.

Inputs	:Vre	- i	Reentry Velocity	m/s
	PhiRe	-	Reentry Flight Path Angle	rad
	BC	-	Ballistic Coefficient	kg/m^2
	H	-	Altitude	km
Outputs	:V	-	Velocity	m/s
	Decl	-	Deceleration	g's
	MaxDecl	-	Maximum deceleration	g's

Locals :

References :

$$v = V_{re}\, e^{\left(\frac{\rho_0 e^{-sclht\, h}}{2\, BC\, sclht\, \mathrm{Sin}\, \phi_{re}}\right)}$$

$$-\frac{\dot{v}}{g}\max = \frac{-\ sclht\ v_{re}^2\, \mathrm{Sin}\, \phi_{re}}{2\, g\, e}$$

$$v_{\max \frac{\dot{v}}{g}} = v_{re}\, e^{-.5} \approx 0.61\, v_{re}$$

$$v_{imp} = v_{re}\, e^{\rho_0/2\ \Delta\ sclht\ \mathrm{Sin}\, \phi_{re}}$$

$$-\dot{v}/g_{imp} = \frac{v_{re}^{\ 2}\rho_0}{2\, g\, \Delta}\ e^{\left[\rho_0/\ \Delta\ sclht\ \mathrm{Sin}\, \phi_{re}\right]}$$

$$h_{\max \frac{\dot{v}}{g}} = \frac{1}{sclht}\, \mathrm{Ln}\left(\frac{-\rho_0}{BC\ sclht\ \mathrm{Sin}\, \phi_{re}}\right)$$

Quartic Root Solutions

Ref : Escobal pg 430-433 Assume the general form of :

$$y = Ax^4 + Bx^3 + Cx^2 + Dx + E$$

Rearrange as

$$y = x^4 + B'x^3 + C'x^2 + D'x + E'$$

$$h = -\frac{B'}{4}$$

$$P = 6h^2 + 3B'h + C'$$

$$Q = 4h^3 + 3B'h^2 + 2C'h + D'$$

IF $Q = 0.0$, let $Z = y^2$ and solve the quadratic for Z $(Z^2 + PZ + R = 0)$

$$\left| \begin{array}{l} y = \sqrt{Z} \\ x_i = y_i + h \end{array} \right.$$

$$R = h^4 + B'h^3 + C'h^2 + D'h + E'$$

$$a = \frac{1}{3}\left(3(P^2 - 4R) - 4P^2\right)$$

$$b = \frac{1}{27}\left(16P^3 - 18P(P^2 - 4R) - 27Q^2\right)$$

$$s = -\frac{2}{3}P$$

$$\Delta = \frac{a^3}{27} + \frac{b^2}{4}$$

IF $\Delta > 0.0$ THEN Use Cardan's Formula, be sure to evaluate negative cube roots with SGN function

$$Z_1 = \sqrt[3]{-\frac{b}{2} + \sqrt{\Delta}} + \sqrt[3]{-\frac{b}{2} - \sqrt{\Delta}}$$

$$\mathrm{Root}_{2i} = \frac{\sqrt{-3}}{2}\left(\sqrt[3]{-\frac{b}{2} + \sqrt{\Delta}} - \sqrt[3]{-\frac{b}{2} - \sqrt{\Delta}}\right)$$

$$\mathrm{Root}_{3i} = -\mathrm{Root}_{2i}$$

IF $\Delta = 0.0$ THEN Make sure to evaluate the negative cube roots using SGN function

$$Z_1 = 2\sqrt[3]{-\frac{b}{2}}$$

$$Z_2 = Z_3 = \sqrt[3]{\frac{b}{2}}$$

IF $\Delta < 0.0$ THEN Use Trigonometric identity

$$E_0 = 2\sqrt{\frac{-a}{3}}$$

$$\mathrm{Cos}\,\phi = \frac{-b}{2\sqrt{\frac{-a^3}{27}}} \qquad \mathrm{Sin}\,\phi = \sqrt{1 - \mathrm{Cos}^2\,\phi}$$

$$Z_1 = E_0 \,\mathrm{Cos}\,\frac{\phi}{3}$$

$$Z_2 = E_0 \,\mathrm{Cos}\left(\frac{\phi}{3} + 120^\circ\right)$$

$$Z_3 = E_0 \,\mathrm{Cos}\left(\frac{\phi}{3} + 240^\circ\right)$$

Find R' as the largest value of $(Z_i + s)$

$$\zeta = \frac{1}{2} (P + R' - \frac{Q}{\sqrt{R'}})$$

$$\beta = \frac{1}{2} (P + R' + \frac{Q}{\sqrt{R'}})$$

Solve the quadratics

$$y^2 + \sqrt{R'}\, y + \zeta$$

$$y^2 + \sqrt{R'}\, y + \beta$$

The Roots are then

$$x_i = y_i + h$$

APPENDIX A

PASCAL SOURCE CODE

TECHNICAL ROUTINES

```
}
UNIT AstroLib;

(*    -------------------------------------------------------------       *)
(*                                                                        *)
(*                     Module - ASTROLIB.PAS                              *)
(*                                                                        *)
(* This file contains fundamental Astrodynamic procedures and functions   *)
(* relating to the time functions.                                        *)
(*                                                                        *)
(*    *****************  NOTICE OF GOVERNMENT ORIGIN  ****************     *)
(*                                                                        *)
(* This software has been developed by an employee of the United States   *)
(* Government at the United States Air Force Academy, and is therefore    *)
(* a work of the United States, and is NOT subject to copyright protection *)
(* under the provisions of 17 U.S.C. 105.  ANY use of this work, or       *)
(* inclusion in other works, must comply with the notice provisions of    *)
(* 17 U.S.C. 403.                                                         *)
(*                                                                        *)
(*    *************************************************************       *)
(*                                                                        *)
(* Current : 30 Jan 91 Capt Dave Vallado       VERSION 3.0                *)
(*                                                                        *)
(* Changes : 28 Jan 91 Capt Dave Vallado                                  *)
(*                     Add algorithm section                              *)
(*           20 Sep 90 Capt Dave Vallado                                  *)
(*                     Update small in elorb/randv/angle etc              *)
(*                     Change to Predict for rtasc and decl               *)
(*           20 Apr 90 Capt Dave Vallado       VERSION 2.0                *)
(*                     Version 2.0                                        *)
(*           16 Nov 89 Capt Dave Vallado                                  *)
(*                     Integrated into one file                           *)
(*           12 Feb 89 Capt Dave Vallado                                  *)
(*                     Standardized format                                *)
(*           28 Sep 88 Capt Dave Vallado                                  *)
(*                     Added HMS and DMS to Rad conversions               *)
(*           30 Aug 88 Capt Dave Vallado                                  *)
(*                     Version 1.0                                        *)
(*                                                                        *)
(*    -------------------------------------------------------------       *)
{
```

```
}
                          INTERFACE
(*   ----------------------------------------------------------------   *)
   Uses Math;

   TYPE
        Str11 = STRING[11];
        Str10 = STRING[10];
        Str3  = STRING[3];
   VAR
        Show    : Char;
        FileOut : TEXT;

    { ------------- Routines for Time calculations -------------- }

  Procedure JulianDay        ( Yr,Mon,D,H,M              : Integer;
                               S                         : Extended;
                               VAR JD                    : Extended );

  Procedure DayOfYr2MDHMS    ( Yr                        : Integer;
                               Days                      : Extended;
                               VAR Mon,D,H,M             : Integer;
                               VAR S                     : Extended );

  Procedure InvJulianDay     ( JD                        : Extended;
                               VAR Yr,Mon,D,H,M          : Integer;
                               VAR S                     : Extended );

  Procedure FindDays         ( Year,Month,Day,Hr,Min     : INTEGER;
                               Sec                       : Extended;
                               VAR Days                  : Extended );

  Function  GSTime           ( JD                        : Extended ): Extended;

  Function  GSTim0           ( Yr                        : Integer  ): Extended;

  Procedure LSTime           ( Lon,JD                    : Extended;
                               VAR LST,GST               : Extended );

  Procedure SunRiseSet       ( JDate,Lat,Lon             : Extended;
                               WhichKind                 : CHAR;
                               VAR UTSunRise, UTSunSet   : Extended );

  Procedure HMStoUT          ( Hr,Min                    : Integer;
                               Sec                       : Extended;
                               VAR UT                    : Extended );

  Procedure UTtoHMS          ( UT                        : Extended;
                               VAR Hr,Min                : Integer;
                               VAR Sec                   : Extended );

  Procedure HMStoRad         ( Hr,Min                    : Integer;
                               Sec                       : Extended;
                               VAR HMS                   : Extended );

  Procedure RadtoHMS         ( HMS                       : Extended;
                               VAR Hr,Min                : Integer;
                               VAR Sec                   : Extended );

  Procedure DMStoRad         ( Deg,Min                   : Integer;
                               Sec                       : Extended;
                               VAR DMS                   : Extended );

  Procedure RadtoDMS         ( DMS                       : Extended;
                               VAR Deg,Min               : Integer;
                               VAR Sec                   : Extended );
{
```

```
}
   { -------- Routines for Technical 2-Body calculations --------- }

 Procedure Site             ( Lat,Alt,Lst                 : Extended;
                              VAR RS,VS                   : Vector    );

 Procedure RVToPOS          ( Rho,Az,El,DRho,DAz,DEl      : Extended;
                              VAR Rhovec,DRhovec          : Vector    );

 Procedure Track            ( Rho,Az,El,DRho,DAz,DEl,
                              Lat,Lst                     : Extended;
                              RS                          : Vector;
                              VAR R,V                     : Vector    );

 Procedure Razel            ( R,V,RS                      : Vector;
                              Lat,Lst                     : Extended;
                              VAR Rho,Az,El,DRho,DAz,DEl: Extended );

 Procedure Elorb            ( R,V                         : Vector;
                              VAR P,A,Ecc,Inc,Omega,Argp,
                              Nu,M,U,L,CapPi              : Extended );

 Procedure RandV            ( P,E,Inc,Omega,Argp,Nu,
                              U,L,CapPi                   : Extended;
                              VAR R,V                     : Vector    );

 Procedure Gibbs            ( R1,R2,R3                    : Vector;
                              VAR V2                      : Vector;
                              VAR Theta                   : Extended;
                              VAR flt                     : Integer   );

 Procedure HerrGibbs        ( R1,R2,R3                    : Vector;
                              JD1,JD2,JD3                 : Extended;
                              VAR V2                      : Vector;
                              VAR Theta                   : Extended;
                              VAR flt                     : Integer   );

 Procedure FindCandS        ( ZNew                        : Extended;
                              VAR CNew,SNew               : Extended );

 Procedure NewtonR          ( E,M                         : Extended;
                              VAR E0,Nu                   : Extended );

 Procedure Kepler           ( Ro,Vo                       : Vector;
                              Time                        : Extended;
                              VAR R,V                     : Vector    );

 Procedure Gauss            ( R1,R2                       : Vector;
                              DM                          : Char;
                              Time                        : Extended;
                              VAR V1,V2                   : Vector    );

 Procedure IJKtoLatLon      ( R                           : Vector;
                              JD                          : Extended;
                              VAR GeoCnLat,Lon            : Extended );

 Procedure Sun              ( JD                          : Extended;
                              VAR RSun                    : Vector;
                              VAR RtAsc,Decl              : Extended );

 Procedure Moon             ( JD                          : Extended;
                              VAR RMoon                   : Vector;
                              VAR RtAsc,Decl              : Extended );

 Procedure PlanetRV         ( PlanetNum                   : Integer;
                              JD                          : Extended;
                              VAR R,V                     : Vector    );

 Function  Geocentric       ( Lat                         : Extended ): Extended;

 Function  InvGeocentric    ( Lat                         : Extended ): Extended;

 Procedure Sight            ( R1,R2                       : Vector;
                              VAR LOS                     : Str3      );

 Procedure Light            ( R                           : Vector;
                              JD                          : Extended;
                              VAR LIT                     : Str3      );

 Procedure OMS2             ( Lat,Lon,Alt,Phi,Az,Speed,JD : Extended;
                              VAR R,V                     : VECTOR    );
{
```

```
}
   { -------------- Routines for ICBM calculations --------------- }

 Procedure RngAz              ( LLat,LLon,TLat,TLon,TOF    : Extended;
                                VAR Range, Az              : Extended );

 Procedure Path               ( LLat, LLon, Range, Az      : Extended;
                                VAR TLat, TLon             : Extended );

 Procedure Trajec             ( LLat,LLon,TLat,TLon,Rbo,Q  : Extended;
                                TypePhi                    : Char;
                                VAR Range,Phi,TOF,Az,ICPhi,
                                    ICVbo,ICRbo            : Extended;
                                VAR VN                     : Vector   );

   { ---------- Routines for Orbit Transfer calculations --------- }

 Procedure Hohmann            ( R1,R3,e1,e3,Nu1,Nu3        : Extended;
                                VAR Deltava,Deltavb,TOF    : Extended );

 Procedure OneTangent         ( R1,R3,e1,e3,Nu1,Nu2,Nu3    : Extended;
                                VAR Deltava,Deltavb,TOF    : Extended );

 Procedure GeneralCoplanar    ( R1,R3,e1,e2,e3,Nu1,Nu2a,Nu2b,Nu3 : Extended;
                                VAR Deltava,Deltavb,TOF          : Extended );

 Procedure Rendezvous         ( Rcs1,Rcs2,PhaseI           : Extended;
                                NumRevs                    : Integer;
                                VAR PhaseF,WaitTime        : Extended );

 Procedure Interplanetary     ( R1,R2,Rbo,Rimpact,Mu1,Mut,Mu2  : Extended;
                                VAR Deltav1,Deltav2,Vbo,Vretro : Extended );

 Procedure Reentry            ( Vre,PhiRe,BC,h             : Extended;
                                VAR V,Decl,MaxDecl         : Extended );

 Procedure HillsR             ( R,V                        : Vector;
                                Alt,t                      : Extended;
                                VAR R1,V1                  : Vector   );

 Procedure HillsV             ( R                          : Vector;
                                Alt,t                      : Extended;
                                VAR V                      : Vector   );

{
```

```
   { ------- Routines for Technical Perturbed calculations ------- }

   { ----------- and Numerical Integration techniques ------------ }

Procedure Target             ( RInt,VInt,RTgt,VTgt        : Vector;
                               Dm                         : CHAR;
                               TOF                        : Extended;
                               VAR V1t,V2t,DV1,DV2        : Vector  );

Procedure PKepler            ( Ro,Vo                      : Vector;
                               DeltaT                     : Extended;
                               VAR R,V                    : Vector    );

Procedure J2DragPert         ( Inc,E,N,NDot               : Extended;
                               VAR OmegaDOT,ArgpDOT,EDOT  : Extended );

Procedure Predict            ( JD,JDEpoch,no,Ndot,Eo,Edot,inco,
                               Omegao,OmegaDot,Argpo,ArgpDot,Mo,
                               Lat,Lon,Alt                : Extended;
                               VAR Rho,Az,El,RtAsc,Decl   : Extended;
                               VAR Vis                    : Str11     );

Procedure Deriv              ( ITime                      : Extended;
                               X                          : Matrix;
                               VAR XDot                   : Matrix    );

Procedure PertAccel          ( R,V                        : Vector;
                               ITime                      : Extended;
                               WhichOne                   : Integer;
                               BC                         : Extended;
                               VAR APert                  : Vector    );

Procedure PDeriv             ( X                          : Matrix;
                               ITime                      : Extended;
                               DerivType                  : Str10;
                               BC                         : Extended;
                               VAR XDot                   : Matrix    );

Procedure RK4                ( ITime                      : Extended;
                               DT                         : Extended;
                               N                          : Integer;
                               DerivType                  : Str10;
                               BC                         : Extended;
                               VAR X                      : Matrix    );

Procedure ATMOS              ( R                          : Vector;
                               VAR Rho                    : Extended );

Procedure CHEBY              ( ALT                        : Extended;
                               VAR PALT,RHOALT            : Extended );

{ ------------------ Constants used in this Library -----------------
|
|   Rad          57.29577951308230     Degrees per radian
|   HalfPi        1.57079632679490
|   Pi            3.14159265358979
|   TwoPi         6.28318530717959
|
|   OmegaEarth    0.0588335906868878   Angular rotation of Earth (Rad/TU)
|   RadPerDay     6.30038809866574     Rads Earth rotates in 1 Solar Day
|   TUMin        13.44685108204        Minutes per Time Unit
|   TUDaySun     54.20765355           days per sun TU
|
|   EESqrd        0.00669437999013     Eccentricity of Earth's shape squared
|   Flat          0.003352810664747352 Flatenning of the Earth
|
|   GMS          332952.9364           Sun Gravitational Parameter DU3/TU2
|   GMM           0.01229997           Moon Gravitational Parameter DU3/TU2
|   J2            0.00108263
|   J3           -0.00000254
|   J4           -0.00000161
|   TUDay         0.00933809102919444  Days per Time Unit
 ------------------------------------------------------------------- }

(*     ---------------------------------------------------------     *)

                          IMPLEMENTATION

(*     ---------------------------------------------------------     *)
{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE JULIANDAY
|
| This procedure finds the Julian date given the Year, Month, Day, and Time.
|   The Julian date is defined by each elapsed day since noon, 1 Jan 4713 BC.
|   Julian dates are measured from this epoch at noon so astronomers
|   observations may be performed on a single "day". The year range is
|   limited since machine routines for 365 days a year and leap years are
|   valid in this range only. This is due to the fact that leap years occur
|   only in years divisible by 4 and centuries whose number is evenly
|   divisible by 400. ( 1900 no, 2000 yes ... )
|
| NOTE: This Algorithm is taken from the 1988 Almanac for Computers,
|   Published by the U.S. Naval Observatory. The algorithm is good for dates
|   between 1 Mar 1900 to 28 Feb 2100 since the last two terms (from the
|   Almanac) are commented out.
|
| Algorithm     : Find the various terms of the expansion
|                 Calculate the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| Inputs        :
|   Yr          - Year                              1900 .. 2100
|   Mon         - Month                                1 .. 12
|   D           - Day                                  1 .. 28,29,30,31
|   H           - Universal Time Hour                  0 .. 23
|   M           - Universal Time Min                   0 .. 59
|   Sec         - Universal Time Sec                 0.0 .. 59.999
|
| Outputs       :
|   JD          - Julian Date                       days from 4713 B.C.
|
| Locals        :
|   Term1       - Temporary Extended value
|   Term2       - Temporary INTEGER value
|   Term3       - Temporary INTEGER value
|   UT          - Universal Time                        days
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
| References    :
|   1988 Almanac for Computers  pg. B2
|   Escobal       pg. 17-19
|   Kaplan        pg. 329-330
|
 ------------------------------------------------------------------------ }

PROCEDURE JulianDay            ( Yr,Mon,D,H,M           : Integer;
                                 S                      : Extended;
                                 VAR JD                 : Extended );
   VAR
       Term2, Term3 : INTEGER;
       Term1, UT    : Extended;
   BEGIN
     TERM1:= 367.0 * Yr;
     TERM2:= TRUNC( (7* (Yr+TRUNC ( (Mon+9)/12) ) ) / 4 );
     TERM3:= TRUNC( 275*Mon / 9 );
     UT:= ( (S/60.0 + M ) / 60.0 + H ) / 24.0;

     JD:= (TERM1-TERM2+TERM3) + D + 1721013.5 + UT { + ***};
   END; { Procedure JulianDay }

{
```

```
}
{ ------------------------------------------------------------------------
|
|                        PROCEDURE DAYOFYR2MDHMS
|
| This procedure converts the day of the year, days, to the month
|   day, hour, minute and second.
|
| Algorithm     : Set up array for the number of days per month
|                 loop through a temp value while the value is < the days
|                 Perform integer conversions to the correct day and month
|                 Convert remainder into H M S using type conversions
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  26 Feb 1990
|
| Inputs        :
|   Yr          - Year                                1900 .. 2100
|   Days        - Julian Day of the year              0.0  .. 366.0
|
| OutPuts       :
|   Mon         - Month                                  1 .. 12
|   D           - Day                                    1 .. 28,29,30,31
|   H           - Hour                                   0 .. 23
|   M           - Minute                                 0 .. 59
|   Sec         - Second                               0.0 .. 59.999
|
| Locals        :
|   Dayofyr     - Day of year
|   Temp        - Temporary Extended values
|   IntTemp     - Temporary Integer value
|   i           - Index
|
| Constants     :
|   LMonth[12]  - Integer Array containing the number of days per month
|
| Coupling      :
|   None.
|
| References    :
|   None.
|
 ------------------------------------------------------------------------ }

PROCEDURE DayOfYr2MDHMS        ( Yr                   : Integer;
                                   Days               : Extended;
                                   VAR Mon,D,H,M      : Integer;
                                   VAR S              : Extended );
   VAR
      Temp               : Extended;
      IntTemp, i, DayOfYr : Integer;
      LMonth             : Array[1..12] of Integer;
   BEGIN
     { -------------- Set up array of days in month  --------------- }
     FOR i:= 1 to 12 DO
       BEGIN
         CASE i OF
           1,3,5,7,8,10,12 : LMonth[i]:= 31;
           4,6,9,11        : LMonth[i]:= 30;
           2               : LMonth[i]:= 28;
         END;  { Case }
       END;

     DayofYr:= TRUNC(Days );

     { ---------------- Find month and Day of month ---------------- }
     IF ( (Yr-1900) MOD 4 ) = 0 THEN
         LMonth[2]:= 29;
     i:= 1;
     IntTemp:= 0;
     WHILE ( DayOfYr > IntTemp + LMonth[i] ) and ( i < 12 ) DO
       BEGIN
         IntTemp:= IntTemp + LMonth[i];
         i:= i+1;
       END;
     Mon:= i;
     D  := DayOfYr - IntTemp;

     { ---------------- Find hours minutes and seconds ------------- }
     Temp:= (Days - DayofYr )*24.0;
     H   := TRUNC( Temp );
     Temp:= (Temp-H ) * 60.0;
     M   := TRUNC( Temp );
     S   := (Temp-M ) *60.0;
   END;  { Procedure DayofYr2MDHMS }
{
```

```
}
{ -----------------------------------------------------------------------
|
|                          PROCEDURE INVJULIANDAY
|
|  This procedure finds the Year, month, day, hour, minute and second
|  given the Julian date.
|
|  Algorithm     : Set up starting values
|                  Find the elapsed days through the year in a loop
|                  Call routine to find each individual value
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  26 Feb 1990
|
|  Inputs        :
|    JD          - Julian Date                          days from 4713 B.C.
|
|  OutPuts       :
|    Yr          - Year                                 1900 .. 2100
|    Mon         - Month                                   1 .. 12
|    D           - Day                                     1 .. 28,29,30,31
|    H           - Hour                                    0 .. 23
|    M           - Minute                                  0 .. 59
|    Sec         - Second                                0.0 .. 59.999
|
|  Locals        :
|    days        - Day of year plus fraction of a day   days
|    Tu          - Julian Centuries from 1 Jan 1900
|    Temp        - Temporary real values
|    LeapYrs     - Number of Leap years from 1900
|
|  Constants     :
|    None.
|
|  Coupling      :
|    DayofYr2MD    Finds Month, day, hour, minute and second given Days and Yr
|
|  References    :
|    1988 Almanac for Computers  pg. B2
|    Escobal       pg. 17-19
|    Kaplan        pg. 329-330
|
 ----------------------------------------------------------------------- }

PROCEDURE INVJULIANDAY          ( JD                   : Extended;
                                  VAR Yr,Mon,D,H,M     : Integer;
                                  VAR S                : Extended );
   VAR
       Days, Tu, Temp  : Extended;
       LeapYrs         : Integer;
   BEGIN
     { -------------- Find Year and Days of the year --------------- }
     Temp:= JD-2415019.5;
     Tu := Temp / 365.25;
     Yr := 1900 + TRUNC( Tu );
     LeapYrs:= TRUNC( ( Yr-1900-1 )/4.0 );
     Days:= Temp - ((Yr-1900)*365.0 + LeapYrs );

     { ----------- Check for case of beginning of a year ----------- }
     IF Days < 1.0 THEN
       BEGIN
         Yr:= Yr - 1;
         LeapYrs:= TRUNC( ( Yr-1900-1 )/4.0 );
         Days:= Temp - ((Yr-1900)*365.0 + LeapYrs );
       END;

     { ---------------- Find remaing data  ------------------------- }
     DayOfYr2MDHMS( Yr,Days, Mon,D,H,M,S );

   END;  { Procedure Inverse mod jd }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                            PROCEDURE FINDDAYS
|
|  This procedure finds the fractional days through a year given the year,
|    month, day, hour, minute and second.
|
|  Algorithm     : Set up array for the number of days per month
|                  Check for a leap year
|                  Loop to find the elapsed days in the year
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  11 Dec 1990
|
|  Inputs        :
|    Yr          - Year                                    1900 .. 2100
|    Mon         - Month                                      1 .. 12
|    D           - Day                                        1 .. 28,29,30,31
|    H           - Hour                                       0 .. 23
|    M           - Minute                                     0 .. 59
|    Sec         - Second                                   0.0 .. 59.999
|
|  OutPuts       :
|    days        - Day of year plus fraction of a day   days
|
|  Locals        :
|    i           - Index
|
|  Constants     :
|    None.
|
|  Coupling      :
|    None.
|
|  References    :
|    None.
|
 --------------------------------------------------------------------------- }

PROCEDURE FindDays              ( Year,Month,Day,Hr,Min    : INTEGER;
                                  Sec                      : Extended;
                                  VAR Days                 : Extended );
   VAR
      i       : BYTE;
      LMonth : ARRAY[1..12] of INTEGER;
   BEGIN
     FOR i:= 1 to 12 DO
       BEGIN
         CASE i OF
           1,3,5,7,8,10,12 : LMonth[i]:= 31;
           4,6,9,11 : LMonth[i]:= 30;
           2 : LMonth[i]:= 28;
         END;  { Case }
       END;
     IF TRUNC( RealMOD( Year-1900,4 ) ) = 0 THEN
         LMonth[2]:= 29;
     i    := 1;
     Days:= 0.0;
     WHILE (i < Month) and ( i < 12 ) DO
       BEGIN
         Days:= Days + LMonth[i];
         i:= i + 1;
       END;
     Days:= Days + Day + Hr/24.0 + Min/1440.0 + Sec/86400.0;
   END;  { Procedure FindDays }
{
```

```
}
{ ----------------------------------------------------------------------
|
|                              FUNCTION GSTIME
|
|  This function finds the Greenwich Sidereal time.  Notice just the integer
|    part of the Julian Date is used for the Julian centuries calculation.
|
|  Algorithm      : Perform expansion calculation to obtain the answer
|                   Check the answer for the correct quadrant and size
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Feb 1989
|
|  Inputs         :
|    JD           - Julian Date                          days from 4713 B.C.
|
|  OutPuts        :
|    GSTime       - Greenwich Sidereal Time              0 to 2Pi rad
|
|  Locals         :
|    Temp         - Temporary variable for reals         rad
|    Tu           - Julian Centuries from 1 Jan 2000
|
|  Constants      :
|    TwoPi        -
|    RadPerDay    - Rads Earth rotates in 1 Solar Day
|
|  Coupling       :
|    RealMOD        Real MOD function
|
|  References     :
|    1989 Astronomical Almanac pg. B6
|    Escobal        pg. 18 - 21
|    Explanatory Supplement pg. 73-75
|    Kaplan         pg. 330-332
|    BMW            pg. 103-104
|
 ---------------------------------------------------------------------- }

FUNCTION GSTime              ( JD                   : Extended ): Extended;
   CONST
     TwoPi    : Extended = 6.28318530717959;
     RadPerDay: Extended = 6.30038809866574;
   VAR
     Temp, Tu : Extended;
   BEGIN
    Tu := ( INT(JD) + 0.5 - 2451545.0 ) / 36525.0;
    Temp:= 1.753368559 + 628.3319705*Tu + 6.770708127E-06*Tu*Tu +
           RadPerDay*( (FRAC( JD )-0.5) );

    { ---------------------- Check quadrants -------------------- }
    Temp:= RealMOD( Temp,TwoPi );
    IF Temp < 0.0 THEN
        Temp:= Temp + TwoPi;

    GSTime:= Temp;

   END; { Function GSTime }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                          FUNCTION GSTIM0
|
| This function finds the Greenwich Sidereal time at the beginning of a year.
|   This formula is derived from the Astronomical Almanac and is good only for
|   0 hr UT, 1 Jan of a year.
|
| Algorithm      : Find the Julian Date Ref 4713 BC
|                  Perform expansion calculation to obtain the answer
|                  Check the answer for the correct quadrant and size
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Feb 1989
|
| Inputs         :
|   Yr           - Year                               1988, 1989, etc.
|
| OutPuts        :
|   GSTim0       - Greenwich Sidereal Time            0 to 2Pi rad
|
| Locals         :
|   JD           - Julian Date                        days from 4713 B.C.
|   Temp         - Temporary variable for Reals       rad
|   Tu           - Julian Centuries from 1 Jan 2000
|
| Constants      :
|   TwoPi          Two times Pi
|
| Coupling       :
|   RealMOD        Real MOD function
|
| References     :
|   1989 Astronomical Almanac pg. B6
|   Escobal        pg. 18 - 21
|   Explanatory Supplement pg. 73-75
|   Kaplan         pg. 330-332
|   BMW            pg. 103-104
|
 --------------------------------------------------------------------------- }

FUNCTION GSTim0              ( Yr                      : Integer ): Extended;
   CONST
     TwoPi : Extended = 6.28318530717959;
   VAR
     JD, Temp, Tu : Extended;
   BEGIN
    JD := 367.0 * Yr - ( TRUNC((7*(Yr+TRUNC(10/12)))/4) ) +
                          ( TRUNC(275/9) ) + 1721014.5;
    Tu := ( INT(JD) + 0.5 - 2451545.0 ) / 36525.0;
    Temp:= 1.753368559 + 628.3319705*Tu + 6.770708127E-06*Tu*Tu;

    { ---------------------- Check quadrants --------------------- }
    Temp:= RealMOD( Temp,TwoPi );
    IF Temp < 0.0 THEN
        Temp:= Temp + TwoPi;

    GSTim0:= Temp;

   END; { Function GSTim0 }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                              PROCEDURE LSTIME
|
|  This procedure finds the Local Sidereal time at a given location.
|
|  Algorithm       : Find GST through the routine
|                    Find LST and check for size and quadrant
|
|  Author          : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
|  Inputs          :
|    Lon           - Site longitude (WEST -)              -2Pi to 2Pi rad
|    JD            - Julian Date                          days from 4713 B.C.
|
|  OutPuts         :
|    LST           - Local Sidereal Time                  0.0 to 2Pi rad
|    GST           - Greenwich Sidereal Time              0.0 to 2Pi rad
|
|  Locals          :
|    None.
|
|  Constants       :
|    TwoPi           Two times Pi
|
|  Coupling        :
|    RealMOD         Real MOD function
|    GSTime          Finds the Greenwich Sidereal Time
|
|  References      :
|    Escobal         pg. 18 - 21
|    Kaplan          pg. 330-332
|    BMW             pg. 99 -100
|
 --------------------------------------------------------------------------- }

PROCEDURE LSTime                ( Lon,JD                     : Extended;
                                  VAR LST,GST                : Extended );
   CONST
      TwoPi : Extended = 6.28318530717959;
   BEGIN
     GST := GSTime( JD );
     LST := Lon + GST;

     { ---------------------- Check quadrants -------------------- }
     LST := RealMOD( LST,TwoPi );
     IF LST < 0.0 THEN
         LST:= Lst + TwoPi;
   END; { Procedure Lstime }

{
```

```
}
{ ------------------------------------------------------------------------
|
|                           PROCEDURE SUNRISESET
|
|  This procedure finds the Universal time for Sunrise and Sunset given the
|    day and site location.  Note the use of degrees and radians since the
|    Almanac presents the algorithm in these units.
|
|  Algorithm     :
|
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  13 Jan 1991
|
|  Inputs        :
|    JDate       - Julian Date                          days from 4713 B.C.
|    Lat         - Site latitude (SOUTH -)              -Pi/2 to Pi/2 rad
|    Lon         - Site longitude (WEST -)               -2Pi to 2Pi rad
|    WhichKind   - Character for which rise/set          'S' 'C' 'N' 'A'
|
|  OutPuts       :
|    UTSunRise   - Universal time of sunrise at lat-lon        hrs
|    UTSunSet    - Universal time of sunset at lat-lon         hrs
|
|  Locals        :
|
|
|  Constants     :
|    Rad           Radians per degree
|    TwoPi
|    Pi
|
|  Coupling      :
|    InvJulianDay  Finds the Yr Da Mn Hr Mi Se from the Julian Date
|    FindDays      Finds the days from 1 Jan of a year
|    ArcSin        Arc sine function
|    ArcCos        Arc cosine function
|
|  References    :
|    Almanac For Computers 1990 pg. B5-B6
|
 ---------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE SUNRISESET           ( JDate,Lat,Lon            : Extended;
                                 WhichKind                : CHAR;
                                 VAR UTSunRise, UTSunSet  : Extended );
   CONST
      Rad    : Extended = 57.29577951308230;
      TwoPi  : Extended = 6.28318530717959;
      Pi     : Extended = 3.14159265358979;
   VAR
      Z,t,m,l,ra,sindelta,delta,h,sec,days : Extended;
      year,month,day,hr,min : Integer;
   BEGIN
     CASE WhichKind OF
       'S' : Z:= (90.0+50.0/60.0 )/Rad;  { Sunrise / set }
       'C' : Z:=  96.0 / Rad;            { Civil }
       'N' : Z:= 102.0 / Rad;            { Nautical }
       'A' : Z:= 108.0 / Rad;            { Astronomical }
     END;  { Case }
     InvJulianDay( JDate,  Year,Month,Day,Hr,Min,Sec );
     FindDays( Year,Month,Day,Hr,Min,Sec,  Days );

     { ------------------------ Sunrise  -------------------------- }
     t := Days + (6.0 - Lon*Rad/15.0)/24.0; {days}
     M := 0.985600*t - 3.289; {Deg}
     L := M + 1.916*Sin( M/Rad ) + 0.020*Sin( 2.0*M/Rad ) + 282.634; {deg}
     L := RealMOD( L,360.0 );
     Ra:= ArcTan( 0.91746*Tan(L/Rad) ); { rad }
     IF Ra < 0.0 THEN
         Ra:= Ra + TwoPi;
     IF (L > 180.0) and (Ra < Pi) THEN
         Ra:= Ra + Pi;
     IF (L < 180.0) and (Ra > Pi) THEN
         Ra:= Ra - Pi;
     SinDelta:= 0.39782*Sin( L/Rad );  { rad }
     Delta:= ArcSin( SinDelta ); {rad }
     H:= ArcCos( (Cos(z) - SinDelta*Sin(Lat)) / (Cos(Delta)*Cos(Lat) ) ); {rad}
     H:= TwoPi - H;
     t:= H*Rad/15.0 + RA*Rad/15.0 - 0.065710*t - 6.622;
     T:= RealMOD( T, 24.0 );

     UTSunRise:= T - Lon*Rad/15.0; {hrs}
     UTSunRise:= RealMOD( UTSunRise, 24.0 );
     IF UTSunRise < 0.0 THEN
         UTSunRise:= 24.0 + UTSunRise;

     { ------------------------ Sunset  --------------------------- }
     t := Days + (18.0 - Lon*Rad/15.0)/24.0; { days }
     M := 0.985600*t - 3.289; {Deg}
     L := M + 1.916*Sin( M/Rad ) + 0.020*Sin( 2.0*M/Rad ) + 282.634; {deg}
     L := RealMOD( L,360.0 );
     Ra:= ArcTan( 0.91746*Tan(L/Rad) ); { rad }
     IF Ra < 0.0 THEN
         Ra:= Ra + TwoPi;
     IF (L > 180.0) and (Ra < Pi) THEN
         Ra:= Ra + Pi;
     IF (L < 180.0) and (Ra > Pi) THEN
         Ra:= Ra - Pi;
     SinDelta:= 0.39782*Sin( L/Rad );  { rad }
     Delta:= ArcSin( SinDelta ); {rad }
     H := ArcCos( (Cos(z) - SinDelta*Sin(Lat)) / (Cos(Delta)*Cos(Lat) ) ); {rad}
     t := H*Rad/15.0 + RA*Rad/15.0 - 0.065710*t - 6.622;
     UTSunSet:= T - Lon*Rad/15.0; {hrs}
     UTSunSet:= RealMOD( UTSunSet, 24.0 );
     IF UTSunSet < 0.0 THEN
         UTSunSet:= 24.0 + utSunSet;
   END;  { Procedure SunRiseSet }
{
```

```
}
{ ----------------------------------------------------------------------------
|
|                          PROCEDURE HMSTOUT
|
| This procedure converts Hours, Minutes and Seconds into Universal Time.
|
| Algorithm     : Calculate the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| InPuts        :
|   Hr          - Hours                                0 .. 24    ex.    2
|   Min         - Minutes                              0 .. 59    ex.   39
|   Sec         - Seconds                            0.0 .. 59.99 ex.  57.29
|
| OutPuts       :
|   UT          - Universal Time                     HrMin.Sec   ex.239.5729
|
| Locals        :
|   None.
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
 --------------------------------------------------------------------------- }

PROCEDURE HMStoUT            ( Hr,Min                    : Integer;
                               Sec                       : Extended;
                               VAR UT                    : Extended );
   BEGIN
     UT:= Hr*100.0 + Min + Sec/100.0;
   END;  { Procedure HMStoUT }

{ ----------------------------------------------------------------------------
|
|                          PROCEDURE UTTOHMS
|
| This procedure converts Universal Time into Hours, minutes and seconds.
|
| Algorithm     : Calculate the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| Inputs        :
|   UT          - Universal Time                     HrMin.Sec   ex.239.5729
|
| OutPuts       :
|   Hr          - Hours                                0 .. 24    ex.    2
|   Min         - Minutes                              0 .. 59    ex.   39
|   Sec         - Seconds                            0.0 .. 59.99 ex.  57.29
|
| Locals        :
|   None.
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
 --------------------------------------------------------------------------- }

PROCEDURE UTtoHMS            ( UT                        : Extended;
                               VAR Hr,Min                : Integer;
                               VAR Sec                   : Extended );
   BEGIN
     Hr := TRUNC( UT/100.0 );
     Min:= TRUNC( UT-Hr*100.0 );
     Sec:= FRAC( UT ) * 100.0;
   END;  { Procedure UTtoHMS }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE HMSTORAD
|
| This procedure converts Hours, minutes and seconds into radians.  Notice
|   the conversion 0.2617 is simply the radian equivalent of 15 degrees.
|
| Algorithm     : Calculate the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| Inputs        :
|   Hr          - Hours                               0 .. 24    ex.   10
|   Min         - Minutes                             0 .. 59    ex.   15
|   Sec         - Seconds                           0.0 .. 59.99 ex.  30.00
|
| OutPuts       :
|   HMS         - Result                               rad       ex. 2.6856253
|
| Locals        :
|   None.
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
 ---------------------------------------------------------------------- }

PROCEDURE HMStoRad              ( Hr,Min                    : Integer;
                                  Sec                       : Extended;
                                  VAR HMS                   : Extended );
   BEGIN
     HMS:= ( Hr + Min/60.0 + Sec/3600.0 )*0.261799387;
   END;  { Procedure HMStoRad }

{ ------------------------------------------------------------------------
|
|                          PROCEDURE RADTOHMS
|
| This procedure converts radians into Hours, minutes and seconds.  Notice
|   the conversion 0.2617 is simply the radian equivalent of 15 degrees.
|
| Algorithm     : Convert incoming radians to hours
|                 Calculate the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| Inputs        :
|   HMS         - Result                               rad       ex. 2.6856253
|
| Outputs       :
|   Hr          - Hours                               0 .. 24    ex.   10
|   Min         - Minutes                             0 .. 59    ex.   15
|   Sec         - Seconds                           0.0 .. 59.99 ex.  30.00
|
| Locals        :
|   None.
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
 ---------------------------------------------------------------------- }

PROCEDURE RadtoHMS              ( HMS                       : Extended;
                                  VAR Hr,Min                : Integer;
                                  VAR Sec                   : Extended );
   BEGIN
     HMS := HMS   / 0.261799387;
     Hr  := TRUNC( HMS   );
     Min:= TRUNC( (HMS  -Hr)*60.0 );
     Sec:= (HMS  -Hr-Min/60.0 ) * 3600.0;
   END;  { Procedure RadtoHMS }
{
```

```
}
{ -----------------------------------------------------------------------
|
|                         PROCEDURE DMSTORAD
|
| This procedure converts Degrees, minutes and seconds into radians.
|
| Algorithm      : Calculate the answer
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109  25 Aug 1988
|
| Inputs         :
|   Deg          - Degrees                          0 .. 360   ex.   98
|   Min          - Minutes                          0 .. 59    ex.   25
|   Sec          - Seconds                        0.0 .. 59.99 ex.  30.00
|
| OutPuts        :
|   DMS          - Result                            rad       ex. 1.717840
|
| Locals         :
|   None.
|
| Constants      :
|   Rad            Degrees per Radian
|
| Coupling       :
|   None.
|
 ---------------------------------------------------------------------- }

PROCEDURE DMStoRad             ( Deg,Min                  : Integer;
                                 Sec                      : Extended;
                                 VAR DMS                  : Extended );
   CONST
      Rad : Extended = 57.29577951308230;
   BEGIN

     DMS:= ( Deg + Min/60.0 + Sec/3600.0 ) / Rad;

   END;  { Procedure DMStoRad }

{ -----------------------------------------------------------------------
|
|                         PROCEDURE RADTODMS
|
| This procedure converts radians into Degrees, minutes and seconds.
|
| Algorithm      : Calculate the answer
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| Inputs         :
|   DMS          - Result                            rad       ex. 1.717840
|
| Outputs        :
|   Deg          - Degrees                          0 .. 360   ex.   98
|   Min          - Minutes                          0 .. 59    ex.   25
|   Sec          - Seconds                        0.0 .. 59.99 ex.  30.00
|
| Locals         :
|   None.
|
| Constants      :
|   Rad            Degrees per Radian
|
| Coupling       :
|   None.
|
 ---------------------------------------------------------------------- }

PROCEDURE RadtoDMS             ( DMS                      : Extended;
                                 VAR Deg,Min              : Integer;
                                 VAR Sec                  : Extended );
   CONST
      Rad   : Extended = 57.29577951308230;
   BEGIN
     DMS:= DMS * Rad;
     Deg:= TRUNC( DMS );
     Min:= TRUNC( (DMS-Deg)*60.0 );
     Sec:= (DHS-Deg-Min/60.0 ) * 3600.0;
   END;  { Procedure RadtoDMS }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE SITE
|
|  This procedure finds the position and velocity vectors for a site.  The
|    answer is returned in the Geocentric Equatorial (IJK) coordinate system.
|
|  Algorithm     : Set up constants
|                  Find x and z values
|                  Find position vector directly
|                  Call cross to find the velocity vector
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
|  Inputs        :
|    Lat         - Geodetic Latitude                    -Pi/2 to Pi/2 rad
|    Alt         - Altitude                             DU
|    LST         - Local Sidereal Time                  -2Pi to 2Pi rad
|
|  OutPuts       :
|    RS          - IJK Site position vector             DU
|    VS          - IJK Site velocity vector             DU/TU
|
|  Locals        :
|    EarthRate   - IJK Earth's rotation rate vector     rad/TU
|    SinLat      - Variable containing  sin( Lat )      rad
|    Temp        - Temporary Extended value
|    x           - x component of site vector           DU
|    z           - z component of site vector           DU
|
|  Constants     :
|    EESqrd      - Eccentricity of Earth's shape sqrd
|    OmegaEarth  - Angular rotation of Earth (Rad/TU)
|
|  Coupling      :
|    Mag           Magnitude of a vector
|    Cross         Cross product of two vectors
|
|  References    :
|    Escobal       pg. 26 - 29  (includes Geocentric Lat formulation also)
|    Kaplan        pg. 334-336
|    BMW           pg. 94 - 98
|
 ------------------------------------------------------------------------ }

PROCEDURE Site                 ( Lat,Alt,Lst              : Extended;
                                   VAR RS,VS               : Vector    );
   CONST
      EESqrd     : Extended = 0.00669437999013;
      OmegaEarth : Extended = 0.0588335906868878;
   VAR
      SinLat, Temp, x, z : Extended;
      EarthRate          : Vector;
   BEGIN
     { -------------------- Initialize values   ------------------- }
     SinLat      := SIN( Lat );
     Earthrate[1]:= 0.0;
     Earthrate[2]:= 0.0;
     Earthrate[3]:= OmegaEarth;

     { --------- Find x and z components of site vector  ---------- }
     Temp  := SQRT( 1.0 - ( EESqrd*SinLat*SinLat ) );
     x     := ( ( 1.0/Temp ) + Alt )*COS( Lat );
     z     := ( ((1.0-EESqrd)/Temp) + Alt )*SinLat;

     { ---------------- Find Site position vector  --------------- }
     RS[1] := x * COS( Lst );
     RS[2] := x * SIN( Lst );
     RS[3] := z;
     MAG( RS );

     { ---------------- Find Site velocity vector  --------------- }
     CROSS( Earthrate,RS,VS );
   END;  { Procedure Site }

{
```

```
}
{ ----------------------------------------------------------------------
|
|                          PROCEDURE RVToPOS
|
|  This procedure finds range and velocity vectors for a satellite from a radar
|    site in the Topocentric Horizon (SEZ) system.
|
|  Algorithm      : Assign temp values to limit number of trig operations
|                   Find SEZ position vector and magnitude directly
|                   Find SEZ velocity vector and magnitude directly
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
|  Inputs         :
|    Rho          - Satellite range from site            DU
|    Az           - Azimuth                              0.0 to 2Pi rad
|    El           - Elevation                            -Pi/2 to Pi/2 rad
|    DRho         - Range Rate                           DU / TU
|    DAz          - Azimuth Rate                         rad / TU
|    DEl          - Elevation rate                       rad / TU
|
|  OutPuts        :
|    RhoVec       - SEZ Satellite range vector           DU
|    DRhoVec      - SEZ Satellite velocity vector        DU / TU
|
|  Locals         :
|    SinEl        - Variable for sin( El )
|    CosEl        - Variable for cos( El )
|    SinAz        - Variable for sin( Az )
|    CosAz        - Variable for cos( Az )
|
|  Constants      :
|    None.
|
|  Coupling       :
|    Mag            Magnitude of a vector
|
|  References     :
|    BMW            pg. 84 - 85
|
 ---------------------------------------------------------------------- }

PROCEDURE RVToPOS              ( Rho,Az,El,DRho,DAz,DEl    : Extended;
                                 VAR Rhovec,DRhovec         : Vector    );
   VAR
       SinEl, CosEl, SinAz, CosAz : Extended;
   BEGIN
     { -------------------- Initialize values  ------------------ }
     SinEl:= SIN(El);
     CosEl:= COS(El);
     SinAz:= SIN(Az);
     CosAz:= COS(Az);

     { ----------------- Form SEZ range vector  ------------------ }
     Rhovec[1] := -Rho*CosEl*CosAz;
     Rhovec[2] :=  Rho*CosEl*SinAz;
     Rhovec[3] :=  Rho*SinEl;
     MAG( RhoVec );

     { ---------------- Form SEZ velocity vector  ---------------- }
     DRhovec[1] := -Drho*CosEl*CosAz + Rho*SinEl*Del*CosAz + Rho*CosEl*SinAz*Daz;
     DRhovec[2] :=  Drho*CosEl*SinAz - Rho*SinEl*Del*SinAz + Rho*CosEl*CosAz*Daz;
     DRhovec[3] :=  Drho*SinEl        + Rho*Del*CosEl;
     MAG( DRhoVec );
   END;  { Procedure RVToPOS }

{
```

```
{ ------------------------------------------------------------------------
|
|                          PROCEDURE Track
|
|  This procedure finds range and velocity vectors in the Geocentric Equatorial
|    (IJK) system given the following input from a radar site.
|
|  Algorithm      : Find constant values
|                   Find SEZ vectors from RVToPOS
|                   Rotate to find IJK vectors
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
|  Inputs         :
|    Rho          - Satellite range from site             DU
|    Az           - Azimuth                               0.0 to 2Pi rad
|    El           - Elevation                             -Pi/2 to Pi/2 rad
|    DRho         - Range Rate                            DU / TU
|    DAz          - Azimuth Rate                          rad / TU
|    DEl          - Elevation rate                        rad / TU
|    Lat          - Geodetic Latitude                     -Pi/2 to Pi/2 rad
|    LST          - Local SideExtended Time               -2Pi to 2Pi rad
|    RS           - IJK Site position vector              DU
|
|  OutPuts        :
|    R            - IJK Satellite position vector         DU
|    V            - IJK Satellite velocity vector         DU / TU
|
|  Locals         :
|    WCrossR      - Cross product result                  DU / TU
|    RhoVec       - SEZ range vector from site            DU
|    DRhoVec      - SEZ velocity vector from site         DU / TU
|    TempVec      - Temporary vector
|    RhoV         - IJK range vector from site            DU
|    DRhoV        - IJK velocity vector from site         DU / TU
|    EarthRate    - IJK Earth's rotation rate vector      rad / TU
|
|  Constants      :
|    HalfPi       -
|    OmegaEarth   - Angular rotation of Earth (Rad/TU)
|
|  Coupling       :
|    RVToPOS        Find R and V from site in Topocentric Horizon (SEZ) system
|    Cross          Cross product of two vectors
|    Addvec         Add two vectors together
|    Rot3           Rotation about the 3rd axis
|    Rot2           Rotation about the 2nd axis
|
|  References     :
|    BMW            pg. 85-89, 100-101
|
 ------------------------------------------------------------------------ }

PROCEDURE Track                 ( Rho,Az,El,DRho,DAz,DEl,
                                  Lat,Lst                   : Extended;
                                  RS                        : Vector;
                                  VAR R,V                   : Vector    );
   CONST
     HalfPi     : Extended = 1.57079632679490;
     OmegaEarth : Extended = 0.0588335906868878;
   VAR
     WCrossR, RhoVec, DRhoVec, TempVec, RhoV, DRhoV, EarthRate : Vector;
   BEGIN
    { -------------------  Initialize values   ------------------- }
    Earthrate[1]:= 0.0;
    Earthrate[2]:= 0.0;
    Earthrate[3]:= OmegaEarth;

    { -----------  Find SEZ range and velocity vectors ------------ }
    RVToPOS( Rho,Az,El,DRho,DAz,DEl,RhoVec,DRhoVec );

    { ------------  Perform SEZ to IJK transformation ------------- }
    ROT2( RhoVec ,Lat-HalfPi, TempVec );
    ROT3( TempVec,    -LST    , RhoV );
    ROT2( DRhoVec,Lat-HalfPi, TempVec );
    ROT3( TempVec,    -LST    , DRhoV );

    { -----------  Find IJK range and velocity vectors ------------ }
    ADDVEC( RhoV,RS,R );
    CROSS ( Earthrate,R ,WCrossR );
    ADDVEC( DRhoV,WCrossR,V );
   END; { Procedure Track }
{
```

```
}
{ -----------------------------------------------------------------------
|
|                         PROCEDURE Razel
|
|  This procedure calculates Range Azimuth and Elevation and their rates given
|    the Geocentric Equatorial (IJK) Position and Velocity vectors.  Notice the
|    value of small as it can affect rate term calculations. (See Example #4)
|
|  Algorithm      : Find constant values
|                   Loop to find range and velocity vectors
|                   Rotate to find SEZ vectors
|                   Use if statments to find Az and El including special cases
|
|  Author         : Capt Dave  '.lado  USAFA/DFAS  719-472-4109  27 Mar 1990
|
|  Inputs         :
|    R            - IJK Position Vector                    DU
|    V            - IJK Velocity Vector                    DU / TU
|    Lat          - Geodetic Latitude                      -Pi/2 to Pi/2 rad
|    LST          - Local Sidereal Time                    -2Pi to Pi rad
|    RS           - IJK Site Position Vector               DU
|
|  Outputs        :
|    Rho          - Satellite Range from site              DUs
|    Az           - Azimuth                                0.0 to 2Pi rad
|    El           - Elevation                              -Pi/2 to Pi/2 rad
|    DRho         - Range Rate                             DU / TU
|    DAz          - Azimuth Rate                           rad / TU
|    DEl          - Elevation rate                         rad / TU
|
|  Locals         :
|    RhoV         - IJK Range Vector from site             DU
|    DRhoV        - IJK Velocity Vector from site          DU / TU
|    RhoVec       - SEZ Range vector from site             DU
|    DRhoVec      - SEZ Velocity vector from site          DU
|    WCrossR      - Cross product result                   DU / TU
|    EarthRate    - IJK Earth's rotation rate vector       rad / TU
|    TempVec      - Temporary vector
|    Temp         - Temporary Extended value
|    Temp1        - Temporary Extended value
|    i            - Index
|
|  Constants      :
|    HalfPi       -
|    Pi           -
|    OmegaEarth   - Angular rotation of Earth (Rad/TU)
|    Small        - Tolerance for roundoff errors
|
|  Coupling       :
|    Mag            Magnitude of a vector
|    Cross          Cross product of two vectors
|    Rot3           Rotation about the 3rd axis
|    Rot2           Rotation about the 2nd axis
|    Atan2          Arc tangent function which also resloves quadrants
|    Dot            Dot product of two vectors
|
|  References     :
|    BMW            pg. 84-89, 100-101
|
 ----------------------------------------------------------------------- }
{
```

```
}
PROCEDURE Razel                  ( R,V,RS                       : Vector;
                                   Lat,Lst                      : Extended;
                                   VAR Rho,Az,El,DRho,DAz,DEl : Extended );
   CONST
      HalfPi     : Extended = 1.57079632679490;
      Pi         : Extended = 3.14159265358979;
      OmegaEarth : Extended = 0.0588335906868878;
      Small      : Extended = 0.000001;
   VAR
      RhoV, DRhoV, RhoVec, DRhoVec, WCrossR, EarthRate, TempVec : Vector;
      Temp, Templ   : Extended;
      i             : Integer;
   BEGIN
     { -------------------- Initialize values   ------------------- }
     Earthrate[1]:= 0.0;
     Earthrate[2]:= 0.0;
     Earthrate[3]:= OmegaEarth;

     { ------- Find IJK range vector from site to satellite  ------ }
     CROSS( EarthRate,R,WCrossR );
     FOR i:=1 to 3 DO
       BEGIN
         RhoV[i]:=  R[i] - RS[i];
         DRhoV[i]:=  V[i] - WCrossR[i];
       END;
     MAG( RhoV );
     Rho:= RhoV[4];

     { ------------ Convert to SEZ for calculations  ------------- }
     ROT3( RhoV ,    LST    ,  TempVec );
     ROT2( TempVec,HalfPi-Lat,   RhoVec    );
     ROT3( DRhoV,    LST    ,  TempVec );
     ROT2( TempVec,HalfPi-Lat,  DRhoVec    );

     { ------------ Calculate Azimuth and Elevation  -------------- }
     Temp:= SQRT( RhoVec[1]*RhoVec[1] + RhoVec[2]*RhoVec[2] );
     IF ABS( RhoVec[2] ) < Small THEN
         IF Temp < Small THEN
           BEGIN
             Templ:= SQRT( DRhoVec[1]*DRhoVec[1] + DRhoVec[2]*DRhoVec[2] );
             Az:= ATAN2( DRhoVec[2]/Templ , -DRhoVec[1]/Templ );
           END
           ELSE
             IF RhoVec[1] > 0.0 THEN
                 Az:= Pi
               ELSE
                 Az:= 0.0
       ELSE
         Az:= ATAN2( RhoVec[2]/Temp , -RhoVec[1]/Temp );

     IF ( Temp < Small ) THEN
         El:= HalfPi
       ELSE
         El:= ATAN2( RhoVec[3]/Rho , Temp/Rho     );

     { ------ Calculate Range, Azimuth and Elevation rates -------- }
     DRho:= DOT( RhoVec,DRhoVec ) / Rho;
     IF ABS( Temp ) > Small THEN
         DAz:= ( DRhoVec[1]*RhoVec[2] - DRhoVec[2]*RhoVec[1] ) / (Temp*Temp)
       ELSE
         DAz:= 0.0;

     IF ABS( Temp ) > 0.000000001 THEN
         DEl:= ( DRhoVec[3] - DRho*SIN( El ) ) / Temp
       ELSE
         DEl:= 0.0;
   END; { Procedure Razel }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                           PROCEDURE Elorb
|
| This procedure finds the classical orbital elements given the Geocentric
|   Equatorial Position and Velocity vectors.  Special cases for equatorial
|   and circular orbits are also handled.  IF elements are Infinite, they
|   are set to 999999.9.  If elements are Undefined, they are set to 999999.1.
|   Be sure to check for these during output!!
|
| Algorithm     : Initialze variables
|                 If the Hbar magnitude exists, continue, otherwise exit and
|                      assign undefined values
|                   Find vectors and values
|                   Determine the type of orbit with IF statements
|                   Find angles depending on the orbit type
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
|
| Inputs        :
|   R           - IJK Position vector                       DU
|   V           - IJK Velocity vector                       DU / TU
|
| Outputs       :
|   P           - Semi-latus rectum                         DU
|   A           - semi-major axis                           DU
|   E           - eccentricity
|   Inc         - inclination                               0.0 to Pi rad
|   Omaga       - Longitude of Ascending Node               0.0 to 2Pi rad
|   Argp        - Argument of Perigee                       0.0 to 2Pi rad
|   Nu          - True anomaly                              0.0 to 2Pi rad
|   M           - Mean anomaly                              0.0 to 2Pi rad
|   U           - Argument of Latitude       (CI)           0.0 to 2Pi rad
|   L           - True Longitude             (CE)           0.0 to 2Pi rad
|   CapPi       - Longitude of Periapsis     (EE)           0.0 to 2Pi rad
|
| Locals        :
|   Hbar        - Angular Momentum H Vector                 DU2 / TU
|   Ebar        - Eccentricity     E Vector
|   Nbar        - Line of Nodes    N Vector
|   c1          - V**2 - u/R
|   RDotV       - R Dot V
|   c3          - Hk unit vector
|   SME         - Specfic Mechanical Energy                 DU2 / TU2
|   i           - index
|   E           - Eccentric Anomaly                         rad
|   D           - Parabolic Eccentric Anomaly               rad
|   F           - Hyperbolic Eccentric Anomaly              rad
|   Temp        - Temporary variable
|   TypeOrbit   - Type of orbit                             EE, EI, CE, CI
|
| Constants     :
|   HalfPi      -
|   Pi          -
|   TwoPi       -
|   Infinite    - Flag for an infintite element
|   Undefined   - Flag for an undefined element
|   Small       - Tolerance for roundoff errors
|
| Coupling      :
|   MAG           Magnitude of a vector
|   CROSS         Cross product of two vectors
|   DOT           DOT product of two vectors
|   ArcCos        Arc Cosine function
|   ArcCosh       Inverse Hyperbolic cosine function
|   Sinh          Hyperbolic Sine function
|   Sgn           -1.0 or 1.0 depending on the sign
|   Angle         Find the angle between two vectors
|
| References    :
|   BMW           pg. 58 - 71, 181-188
|   Escobal       pg. 104-107
|   Kaplan        pg. 29 - 37
|
 ------------------------------------------------------------------------ }
{
```

```
}
PROCEDURE Elorb                    ( R,V                         : Vector;
                                     VAR P,A,Ecc,Inc,Omega,Argp,
                                     Nu,M,U,L,CapPi              : Extended );
   CONST
      HalfPi    : Extended = 1.57079632679490;
      Pi        : Extended = 3.14159265358979;
      TwoPi     : Extended = 6.28318530717959;
      Small     : Extended = 0.000001;
      Infinite  : Extended = 999999.9;
      Undefined: Extended = 999999.1;
   VAR
      c1, RDotV, c3, SME, Temp, F, D, E  : Extended;
      Hbar, Ebar, Nbar                   : Vector;
      i                                  : Integer;
      TypeOrbit                          : String[2];
   BEGIN
     { --------------------  Initialize values   ------------------- }
     MAG( R );
     MAG( V );

     { ------------------  Find H N and E vectors   --------------- }
     CROSS( R,V,Hbar );

     IF HBar[4] > Small THEN
       BEGIN
         Nbar[1]:= -Hbar[2];
         Nbar[2]:=  Hbar[1];
         Nbar[3]:=   0.0;
         MAG( Nbar );
         c1 := V[4]*V[4] - 1.0/R[4];
         RDotV:= DOT( R,V );
         FOR i:= 1 to 3 DO
             Ebar[i]:= c1*R[i] - RDotV*V[i];
         MAG( Ebar );


     { ---------------  Find a e and semi-latus rectum   ---------- }
         SME:= ( V[4]*V[4]/2.0 ) - ( 1.0/R[4] );
         IF ABS( SME ) > Small THEN
             A:= -1.0 / (2.0*SME)
           ELSE
             A:= Infinite;        { Parabola }
         Ecc:= Ebar[4];
         P := HBar[4]*HBar[4];


     { --------------------  Find inclination   -------------------- }
         c3:= HBar[3]/HBar[4];
         IF ABS( ABS(c3) - 1.0 ) < Small THEN
             IF ABS(HBar[3]) > 0.0 THEN
             { ----------  Equatorial Orbits   ------------- }
                 c3:= SGN(HBar[3]) * 1.0;
         Inc:= ARCCOS( c3 );


     { ---------  Determine type of orbit for later use  ----------- }

           { --- Elliptical, Parabolic, Hyperbolic Inclined --- }
         TypeOrbit:= 'EI';

         IF Ecc < Small THEN
           { -------------  Circular Equatorial  -------------- }
             IF ( Inc < Small ) or ( ABS(Inc-Pi) < Small ) THEN
                 TypeOrbit:= 'CE'
               ELSE
           { -------------  Circular Inclined    -------------- }
                 TypeOrbit:= 'CI'
           ELSE
           { -- Elliptical, Parabolic, Hyperbolic Equatorial -- }
             IF ( Inc < Small ) or ( ABS(Inc-Pi) < Small ) THEN
               TypeOrbit:= 'EE';
{
```

```
}
   { ------------ Find Longitude of Ascending Node -------------- }
      IF NBar[4] > Small THEN
        BEGIN
          Temp:= Nbar[1] / NBar[4];
          IF ABS(Temp) > 1.0 THEN
              Temp:= SGN(Temp) * 1.0;
          Omega:= ARCCOS( Temp );
          IF NBar[2] < 0.0 THEN
              Omega:= TwoPi - Omega;
        END
        ELSE
          Omega:= Undefined;

   { ----------------- Find Argument of perigee ----------------- }
      IF TypeOrbit = 'EI' THEN
        BEGIN
          ANGLE( Nbar,Ebar,  Argp );
          IF EBar[3] < 0.0 THEN
              Argp:= TwoPi - Argp;
        END
        ELSE
          Argp:= Undefined;

   { ------------- Find True Anomaly at Epoch    --------------- }
      IF TypeOrbit[1] = 'E' THEN
        BEGIN
          ANGLE( Ebar,r,  Nu );
          IF RDotV < 0.0 THEN
              Nu:= TwoPi - Nu;
        END
        ELSE
          Nu:= Undefined;

   { -----  Find Argument of Latitude - Circular Inclined  ------ }
      IF TypeOrbit = 'CI' THEN
        BEGIN
          ANGLE( NBar,R,  U );
          IF R[3] < 0.0 THEN
              U:= TwoPi - U;
        END
        ELSE
          U:= Undefined;

   { ---- Find Longitude of Perigee - Elliptical Equatorial ----- }
      IF ( EBar[4] > Small ) and ( TypeOrbit = 'EE' ) THEN
        BEGIN
          Temp:= Ebar[1]/EBar[4];
          IF ABS(Temp) > 1.0 THEN
              Temp:= SGN(Temp) * 1.0;
          CapPi:= ARCCOS( Temp );
          IF Ebar[2] < 0.0 THEN
              CapPi:= TwoPi - CapPi;
          IF Inc > HalfPi THEN
              CapPi:= TwoPi - CapPi;
        END
        ELSE
          CapPi:= Undefined;

   { -------- Find True Longitude - Circular Equatorial --------- }
      IF ( R[4] > Small ) and ( TypeOrbit = 'CE' ) THEN
        BEGIN
          Temp:= R[1]/R[4];
          IF ABS(Temp) > 1.0 THEN
              Temp:= SGN(Temp) * 1.0;
          L:= ARCCOS( Temp );
          IF R[2] < 0.0 THEN
              L:= TwoPi - L;
          IF Inc > HalfPi THEN
              L:= TwoPi - L;
        END
        ELSE
          L:= Undefined;

 {
```

```
}
    { ------------- Find Mean Anomaly for all orbits ------------- }
    { ------------------------ Hyperbolic  ----------------------- }
        IF (Ecc-1.0) > Small THEN
          BEGIN
            F:= ArcCosH( (Ecc+Cos(Nu)) / (1.0+Ecc*Cos(Nu)) );
            M:= Ecc*Sinh( F ) - F;
          END
          ELSE

    { ------------------------ Parabolic ------------------------- }
            IF ABS( Ecc-1.0 ) < Small THEN
              BEGIN
                D:= SQRT( p ) * Tan( Nu );
                M:= (1.0/6.0)*( 3.0*p*D + D*D*D );
              END
              ELSE

    { ------------------------ Elliptical ------------------------ }
                IF Ecc > Small THEN
                  BEGIN
                    Temp:= 1.0 + Ecc*Cos(Nu);
                    IF ABS(Temp) < Small THEN
                        M:= 0.0
                      ELSE
                      BEGIN
                        c1:= ( SQRT(1.0-Ecc*Ecc)*Sin(Nu) ) / Temp;
                        c3:= ( Ecc+Cos(Nu) ) / Temp;
                        IF ABS(c1) > 1.0 THEN
                            c1:= SGN(c1) * 1.0;
                        IF ABS(c3) > 1.0 THEN
                            c3:= SGN(c3) * 1.0;
                        E:= ATan2( c1,c3 );
                        M:= E - Ecc*Sin( E );
                      END;
                  END
                  ELSE

    { ------------------------- Circular ------------------------- }
                    IF TypeOrbit = 'CE' THEN
                          M:= L
                        ELSE
                          M:= U;

        M:= REALMOD( M,TwoPi );
        IF M < 0.0 THEN
            M:= M + TwoPi;

{}      IF Show = 'Y' THEN
{}        BEGIN
{}          WriteLn( 'H = ':6,Hbar[1]:13:7,Hbar[2]:14:7,Hbar[3]:14:7,Hbar[4]:14:7 );
{}          WriteLn( 'N = ':6,Nbar[1]:13:7,Nbar[2]:14:7,Nbar[3]:14:7,Nbar[4]:14:7 );
{}          WriteLn( 'E = ':6,Ebar[1]:13:7,Ebar[2]:14:7,Ebar[3]:14:7,Ebar[4]:14:7 );
{}          WriteLn( 'SME=':6,SME:13:7,' DU2/TU2' );
{}          WriteLn( 'TypeOrbit = ',TypeOrbit:3 );
{}        END;
{}      IF Show = 'S' THEN
{}        BEGIN
{}          WriteLn( FileOut,'H = ':6,Hbar[1]:13:7,Hbar[2]:14:7,Hbar[3]:14:7,Hbar[4]:14:7
{}          WriteLn( FileOut,'N = ':6,Nbar[1]:13:7,Nbar[2]:14:7,Nbar[3]:14:7,Nbar[4]:14:7
{}          WriteLn( FileOut,'E = ':6,Ebar[1]:13:7,Ebar[2]:14:7,Ebar[3]:14:7,Ebar[4]:14:7
{}          WriteLn( FileOut,'SME=':6,SME:13:7,' DU2/TU2' );
{}          WriteLn( FileOut,'TypeOrbit = ',TypeOrbit:3 );
{}        END;

      END  { If Hbar[4] > 0 the orbit is possible }
      ELSE
      BEGIN
        P     := Undefined;
        A     := Undefined;
        Ecc   := Undefined;
        Inc   := Undefined;
        Omega := Undefined;
        Argp  := Undefined;
        Nu    := Undefined;
        M     := Undefined;
        U     := Undefined;
        l     := Undefined;
        CapPi := Undefined;
      END;

   END; { Procedure Elorb }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                              PROCEDURE RandV
|
|  This procedure finds the position and velocity vectors in Geocentric
|    Equatorial (IJK) system given the classical orbit elements.  NOTICE
|    P is used for calculations and that semi-major axis a, is not.  This
|    convention allows parabolic orbits to be treated as well as the other
|    conic sections.  Notice the special cases leave Argp, Omega and Nu equal
|    to zero, rather than setting them to some large number as a flag for
|    infinite or undefined.  This allows the routine to process different types
|    of orbits with ONE transformation matrix since zeros will leave the vectors
|    unchanged during that phase of the transformation.
|
|  Algorithm     : Select the type of orbit through IF statements
|                      and assign Omega, Argp, and Nu
|                      Although these values change, they are NOT passed back
|                  Find the PQW position and veocity vectors
|                  Rotate by 3-1-3 to IJK.  Order is important
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
|
|  Inputs        :
|    P           - Semi-latus rectum                        DU
|    E           - eccentricity                             0.0 to ...
|    Inc         - inclination                              0.0 to Pi  rad
|    Omega       - Longitude of Ascending Node              0.0 to 2Pi rad
|    Argp        - Argument of Perigee                      0.0 to 2Pi rad
|    Nu          - True anomaly                             0.0 to 2Pi rad
|    U           - Argument of Latitude        (CI)         0.0 to 2Pi rad
|    L           - True Longitude              (CE)         0.0 to 2Pi rad
|    CapPi       - Longitude of Periapsis      (EE)         0.0 to 2Pi rad
|
|  Outputs       :
|    R           - IJK Position vector                      DU
|    V           - IJK Velocity vector                      DU / TU
|
|  Locals        :
|    Temp        - Temporary Extended value
|    Rpqw        - PQW Position vector                      DU
|    Vpqw        - PQW Velocity vector                      DU / TU
|    TempVec     - PQW Velocity vector
|
|  Constants     :
|    Pi
|    Small       - Tolerance for roundoff errors
|
|  Coupling      :
|    MAG           Magnitude of a vector
|    ROT3          Rotation about the 3rd axis
|    ROT1          Rotation about the 1st axis
|
|  References    :
|    BMW           pg. 71-73, 80-83
|    Escobal       pg. 68-83
|
 --------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE RandV                ( P,E,Inc,Omega,Argp,Nu,
                                 U,L,CapPi               : Extended;
                                 VAR R,V                 : Vector   );
   CONST
      Pi    : Extended = 3.14159265358979;
      Small: Extended = 0.000001;
   VAR
      Temp                 : Extended;
      Rpqw, Vpqw, TempVec : Vector;
   BEGIN
     { ------------------------------------------------------------
     |  Determine what type of orbit is involved and set up the
     |  set up angles for the special cases.
      ------------------------------------------------------------ }
     IF E < SMALL THEN
             { ------------ Circular Equatorial  ---------- }
         IF ( Inc < Small ) or ( ABS(Inc - Pi) < Small ) THEN
           BEGIN
             Argp  := 0.0;
             Omega:= 0.0;
             Nu    := L;
           END
           ELSE
             { ---------- Circular Inclined  -------------- }
           BEGIN
             Argp:= 0.0;
             Nu   := U;
           END
       ELSE
             { ---------- Elliptical Equatorial  ---------- }
         IF ( Inc < Small ) or ( ABS(Inc - Pi) < Small ) THEN
           BEGIN
             Argp  := Cappi;
             Omega:= 0.0;
           END;

     { ---------- Form PQW position and velocity vectors ---------- }
     Temp:= P / (1.0 + E*COS(NU));
     Rpqw[1]:= Temp*COS(NU);
     Rpqw[2]:= Temp*SIN(NU);
     Rpqw[3]:=      0.0;
     Vpqw[1]:=     -SIN(NU)     / SQRT(P);
     Vpqw[2]:=   (E + COS(NU)) / SQRT(P);
     Vpqw[3]:=       0.0;
     MAG( Rpqw );
     MAG( Vpqw );

     { -------------- Perform transformation to IJK  -------------- }
     ROT3( Rpqw    , -Argp , TempVec );
     ROT1( TempVec, -Inc   , TempVec );
     ROT3( TempVec, -Omega,    R      );

     ROT3( Vpqw    , -Argp , TempVec );
     ROT1( TempVec, -Inc   , TempVec );
     ROT3( TempVec, -Omega,    V      );
   END; { Procedure RandV }

{
```

```
}
{ ----------------------------------------------------------------------------
|
|                          PROCEDURE GIBBS
|
| This procedure performs the Gibbs method of orbit determination.  This
|   method determines the velocity at the middle point of the 3 given position
|   vectors.  Several flags are passed back.
|
|                 Flt = 0 ok
|                 Flt = 1 not coplanar
|                 Flt = 2 orbit impossible
|
| The Gibbs method is best suited for coplanar, sequential position vectors
|   which are more than about 10 deg apart.  Notice the angle is passed back
|   so the user may make a decision about the accuracy of the calculations as
|   vectors which are 120 deg apart may be accurate, while vectors 8 deg
|   apart would not.  The method will calculate the resulting velocity using
|   the vectors IN THE ORDER GIVEN.  IF the calculations are not possible,
|   V2 is set to 0.0.  Notice a 1 deg tolerance is allowed for the coplanar
|   check.  This is necessary to allow for noisy data in the estimation project.
|
| Algorithm     : Initialize values including the answer
|                 Find if the vectors are coplanar, else set a flag
|                 Check that the orbit is possible, else set a flag
|                 Find the largest angle between the vectors
|                 Calculate the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   28 Mar 1990
|
| Inputs        :
|   R1          - IJK Position vector #1              DU
|   R2          - IJK Position vector #2              DU
|   R3          - IJK Position vector #3              DU
|
| OutPuts       :
|   V2          - IJK Velocity Vector for R2          DU / TU
|   Theta       - Angle between vectors               rad
|   Flt         - Flag indicating success             0, 1, 2
|
| Locals        :
|   tover2      -
|   l           -
|   Small       - Tolerance for roundoff errors
|   r1mr2       - Magnitude of r1 - r2
|   r3mr1       - Magnitude of r3 - r1
|   r2mr3       - Magnitude of r2 - r3
|   p           - P Vector     r2 x r3
|   q           - Q Vector     r3 x r1
|   w           - W Vector     r1 x r2
|   d           - D Vector     p + q + w
|   n           - N Vector (r1)p + (r2)q + (r3)w
|   s           - S Vector (r2-r3)r1+(r3-r1)r2+(r1-r2)r3
|   b           - B Vector     d x r2
|   Theta1      - Temporary angle between the vectors    rad
|   PN          - P Unit Vector
|   R1N         - R1 Unit Vector
|   dn          - D Unit Vector
|   nn          - N Unit Vector
|   i           - index
|
| Constants     :
|   None.
|
| Coupling      :
|   MAG           Magnitude of a vector
|   CROSS         Cross product of two vectors
|   DOT           Dot product of two vectors
|   ADD3VEC       Add three vectors
|   LNCOM2        Multiply two vectors by two constants
|   LNCOM3        Add three vectors each multiplied by a constant
|   NORM          Creates a Unit Vector
|   ANGLE         Angle between two vectors
|
| References    :
|   BMW           pg. 109-116
|   Escobal       pg. 306-307
|
 ---------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE GIBBS                  ( R1,R2,R3                   : Vector;
                                   VAR V2                     : Vector;
                                   VAR Theta                  : Extended;
                                   VAR flt                    : Integer  );
   VAR
      tover2, l, Small, r1mr2, r3mr1, r2mr3, Theta1        : Extended;
      p, q, w, d, n, s, b, Pn, R1N,Dn,nn                   : Vector;
      i                                                    : Integer;
   BEGIN
     { -------------------- Initialize values   ------------------- }
     Small:= 0.000001;  Theta:= 0.0;   Flt := 0;
     Mag( R1 );         Mag( R2 );     Mag( R3 );
     FOR i:= 1 to 4 DO
         V2[i]:= 0.0;

     { -----------------------------------------------------------
     |  Determine if the vectors are coplanar.  The DOT product of R1 and the
     |    normal vector of R2 and R3 will be 0 if all three vectors are coplanar.
     |    The Vectors are normalized to accept very small and very large
     |    vectors.  The magnitudes are left out of the DOT product equation     :
     |    r1n dot pn = r1n pn Cos()      : since each vector is normalized, so the
     |    magnitudes are 1.0.  A 1 deg tolerance is allowed for estimation, and
     |    is implemented by allowing the angle between R1n and Pn to range from
     |    89.0 to 91.0 deg, or Cos(89.0) = 0.017452406.
       ----------------------------------------------------------- }
     CROSS( R2,R3,P );
     CROSS( R3,R1,Q );
     CROSS( R1,R2,W );
     NORM( P,PN );
     NORM( R1,R1N );
     IF ABS( DOT(R1N,PN) ) > 0.017452406 THEN { Not coplanar }
         Flt:= 1
       ELSE
         BEGIN
           ADD3VEC( P,Q,W,D );
           LNCOM3( R1[4],R2[4],R3[4],P,Q,W,N );
           NORM( N,NN );
           NORM( D,DN );

      { ----------------------------------------------------------
      |  Determine if the orbit is possible.  Both D and N must be in
      |    the same direction, and non-zero.
        ---------------------------------------------------------- }
           IF ( ABS(d[4])<Small ) or ( ABS(n[4])<Small ) or
              ( Dot(nn,dn) < Small ) THEN
              Flt:= 2              { Orbit not possible }
            ELSE
              BEGIN
                Angle( R1,R2, Theta );
                Angle( R2,R3, Theta1 );
                IF Theta1 > Theta THEN
                    Theta:= Theta1;

      { ------------ Perform Gibbs method to find V2  ------------- }
                R1mr2:= R1[4]-R2[4];
                R3mr1:= R3[4]-R1[4];
                R2mr3:= R2[4]-R3[4];
                LNCOM3(R1mr2,R3mr1,R2mr3,R3,R2,R1,S);
                CROSS( d,r2,b );
                L    := 1.0 / Sqrt(d[4]*n[4]);
                Tover2:= L / R2[4];
                LNCOM2(Tover2,L,B,S,V2);
              END;
         END;

{}   IF ( Show = 'Y') and ( Flt = 0 ) THEN
{}     BEGIN
{}       WriteLn( 'P vector = ':16,P[1]:9:3,P[2]:9:3,P[3]:9:3 );
{}       WriteLn( 'Q vector = ':16,Q[1]:9:3,Q[2]:9:3,Q[3]:9:3 );
{}       WriteLn( 'W vector = ':16,W[1]:9:3,W[2]:9:3,W[3]:9:3 );
{}       WriteLn( 'D vector = ':16,D[1]:9:3,D[2]:9:3,D[3]:9:3 );
{}       WriteLn( 'N vector = ':16,N[1]:9:3,N[2]:9:3,N[3]:9:3 );
{}       WriteLn( 'S vector = ':16,S[1]:9:3,S[2]:9:3,S[3]:9:3 );
{}       WriteLn( 'B vector = ':16,B[1]:9:3,B[2]:9:3,B[3]:9:3 );
{}     END;

   END;  { Procedure Gibbs }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                        PROCEDURE HERRGIBBS
|
| This procedure implements the Herrick-Gibbs approximation for orbit
|   determination, and finds the middle velocity vector for the 3 given
|   position vectors.  The method is good for fast calculations and small
|   angles, <= 10 deg.  Notice the angle is passed back since vectors which
|   are 12 deg apart may actually be accurate, while vectors which are 170 deg
|   apart would not.  The observations MUST be sequential and taken on one
|   revolution.  The Use of Julian Dates for input makes it much easier to
|   perform calculations where the sights occur around midnight.  Several
|   flags are passed back:
|
|          Flt = 0 ok
|          Flt = 1 orbits not coplanar
|          Flt = 2 angles bewteen the vectors are larger than 10 deg
|
|   Notice a 1 deg tolerance is allowed for the coplanar check.  This is
|   necessary to allow for noisy data in the estimation project.
|
| Algorithm      : Initialize values including the answer
|                  Find if the vectors are coplanar, else set a flag
|                  Find the largest angle between the vectors
|                  Calculate the Taylor series for the answer
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109  28 Mar 1990
|
| Inputs         :
|   R1           - IJK Position vector #1                DU
|   R2           - IJK Position vector #2                DU
|   R3           - IJK Position vector #3                DU
|   JD1          - Julian Date of 1st sighting           days from 4713 B.C.
|   JD2          - Julian Date of 2nd sighting           days from 4713 B.C.
|   JD3          - Julian Date of 3rd sighting           days from 4713 B.C.
|
| OutPuts        :
|   V2           - IJK Velocity Vector for R2            DU / TU
|   Theta        - Angle between vectors                 rad
|   Flt          - Flag indicating success               0, 1, 2
|
| Locals         :
|   dt21         - time delta between r1 and r2          TU
|   dt31         - time delta between r3 and r1          TU
|   dt32         - time delta between r3 and r2          TU
|   p            - P vector    r2 x r3
|   PN           - P Unit Vector
|   R1N          - R1 Unit Vector
|   Theta1       - temporary Angle between vectors       rad
|   TolAngle     - Tolerance angle  (10 deg)             rad
|   Term1        - First Term for HGibbs expansion
|   Term2        - Second Term for HGibbs expansion
|   Term3        - Third Term for HGibbs expansion
|   i            - Index
|
| Constants      :
|   TUMin        - Minutes in each Time Unit             13.44685108204
|
| Coupling       :
|   MAG            Magnitude of a vector
|   CROSS          Cross product of two vectors
|   DOT            Dot product of two vectors
|   ArcCos         Arc Cosine function
|   NORM           Creates a Unit Vector
|   LNCOM3         Combination of three scalars and three vectors
|   ANGLE          Angle between two vectors
|
| References     :
|   Escobal        pg. 254-256, 304-306
|
 ------------------------------------------------------------------------ }
{
```

```
}
PROCEDURE HerrGibbs             ( R1,R2,R3                : Vector;
                                  JD1,JD2,JD3             : Extended;
                                  VAR V2                  : Vector;
                                  VAR Theta               : Extended;
                                  VAR Flt                 : Integer  );
   CONST
     TUMin : Extended = 13.44685108204;
   VAR
     dt21, dt31, dt32, Term1,Term2,Term3,Theta1,TolAngle: Extended;
     p, Pn, R1n    : Vector;
     i             : Integer;
  BEGIN
    { --------------------  Initialize values   ------------------- }
    Flt  :=  0;
    Theta:=  0.0;
    Mag( R1 );
    Mag( R2 );
    Mag( R3 );
    FOR i:= 1 to 4 DO
        V2[i]:= 0.0;
    TolAngle:= 0.174532925;
    DT21:= (JD2-JD1)*1440.0/TUMin;
    DT31:= (JD3-JD1)*1440.0/TUMin;    { differences in times }
    DT32:= (JD3-JD2)*1440.0/TUMin;

    { ------------------------------------------------------------
    |  Determine if the vectors are coplanar.  The DOT product of R1 and the
    |    normal vector of R2 and R3 will be 0 if all three vectors are coplanar.
    |    The Vectors are normalized to accept very small and very large
    |    vectors.  The magnitudes are left out of the DOT product equation      :
    |    r1n dot pn = r1n pn Cos()       : since each vector is normalized, so the
    |    magnitudes are 1.0.  A 1 deg tolerance is allowed for estimation, and
    |    is implemented by allowing the angle between R1n and Pn to range from
    |    89.0 to 91.0 deg, or Cos(89.0) = 0.017452406.
     ------------------------------------------------------------ }
    CROSS( R2,R3,P );
    NORM( P,PN );
    NORM( R1,R1N );
    IF ABS( DOT(R1N,PN) ) > 0.017452406 THEN { Not coplanar }
       Flt:= 1
      ELSE
       BEGIN
    { ------------------------------------------------------------
    | Check the size of the angles between the three position vectors.
    |   Herrick Gibbs only gives "reasonable" answers when the
    |   position vectors are reasonably close.  10 deg is only an estimate.
     ------------------------------------------------------------ }
         Angle( R1,R2, Theta );
         Angle( R2,R3, Theta1 );
         IF Theta1 > Theta THEN
             Theta:= Theta1;
         IF Theta > TolAngle THEN
             Flt:= 2;

    { ------------- Perform Herrick-Gibbs method to find V2 ------- }

         Term1:= -dt32*( 1.0/(dt21*dt31) + 1.0/(12*r1[4]*r1[4]*r1[4]) );
         Term2:= (dt32-dt21)*( 1.0/(dt21*dt32) + 1.0/(12*r2[4]*r2[4]*r2[4]) );
         Term3:=  dt21*( 1.0/(dt32*dt31) + 1.0/(12*r3[4]*r3[4]*r3[4]) );
         LNCOM3( Term1,Term2,Term3,R1,R2,R3, V2 );
       END;  { if not coplanar }

   END; { Procedure HerrGibbs }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                           PROCEDURE FINDCandS
|
|  This procedure calculates the C and S functions for use in the Universal
|    Variable calculations.  NOTE equality is handled by the series expansion
|    terms to eliminate potential discontinuities.  The series is only used for
|    negative values of Z since the truncation results in rather large errors
|    as Z gets larger than about 10.0.
|
|  Algorithm     : If Z is greater than zero, use the exact formulae else
|                    use the series form
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  30 Jan 1991
|
|  Inputs        :
|    ZNew        - Z variable
|
|  Outputs       :
|    CNew        - C function value
|    SNew        - S function value
|
|  Locals        :
|    ZNewSqrd    - ZNew squared
|    ZNewFrth    - ZNew to the fourth power
|    SqrtZ       - Square root of ZNew
|
|  Constants     :
|    None.
|
|  Coupling      :
|    None.
|
|  References    :
|    BMW           pg. 207-210   (Complete graph of S and C)
|    Kaplan        pg. 304-305
|
 ---------------------------------------------------------------------------- }

PROCEDURE FindCandS             ( ZNew                  : Extended;
                                  VAR CNew,SNew         : Extended );
   VAR
     ZNewFrth,ZNewSqrd, SqrtZ : Extended;
   BEGIN
     IF ZNew > 0.0 THEN
       BEGIN
         SqrtZ := SQRT( Znew );
         CNew  := (1.0-COS( SqrtZ )) / ZNew;
         SNew  := (SqrtZ-SIN( SqrtZ )) / ( SqrtZ*SqrtZ*SqrtZ );
       END
       ELSE
       BEGIN
         ZNewSqrd := ZNew*ZNew;
         ZNewFrth := ZNewSqrd*ZNewSqrd;
         CNew     :=   0.5   - ZNew/24.0  + ZNewSqrd/720.0  - (ZNewSqrd*ZNew)/40320.0
                    + ZNewFrth/3628800.0 - ZNewFrth*ZNew/479001600.0;
         SNew     := 1.0/6.0 - ZNew/120.0 + ZNewSqrd/5040.0 - (ZNewSqrd*ZNew)/362880.0
                    + ZNewFrth/39916800.0 - ZNewFrth*ZNew/6227020800.0;
       END;
   END;  { Procedure FindCandS }
{
```

```
}
{ ---------------------------------------- ------------------------------
|
|                         PROCEDURE NEWTONR
|
| This procedure performs the Newton Rhapson iteration to find the
|   Eccentric Anomaly given the Mean anomaly.  The True Anomaly is also
|   calculated.
|
| Algorithm     : Setup the first guess
|                 Loop while the answer has not converged
|                 Write an error if the answer doesn't converge
|                 Find the True Anomaly using ATAN2 to resolve quadrants
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| Inputs        :
|   e           - Eccentricity                          0.0 - 1.0
|   M           - Mean Anomaly                          0.0 - 2Pi rad
|
| Outputs       :
|   E0          - Eccentric Anomaly                     0.0 - 2Pi rad
|   Nu          - True Anomaly                          0.0 - 2Pi rad
|
| Locals        :
|   E1          - Eccentric Anomaly, next value         rad
|   Sinv        - Sine of Nu
|   Cosv        - Cosine of Nu
|   Ktr         - Index
|
| Constants     :
|   None.
|
| Coupling      :
|   Atan2         Arc tangent function which also resloves quadrants
|
| References    :
|   BMW           pg. 184-186, 220-222
|
 ---------------------------------------------------------------------------- }

PROCEDURE NewtonR              ( E,M                     : Extended;
                                 VAR E0,Nu               : Extended );
   VAR
      E1, Sinv, Cosv : Extended;
      Ktr            : INTEGER;
   BEGIN
     { -------------------- Initialize values   ------------------- }
     E0  := M;
     Ktr := 1;

     { --------- Newton Iteration for Eccentric Anomaly ----------- }
     E1:= E0 - ( ( E0 - e*SIN(E0)-m ) / ( 1.0 - e*COS(E0) ) );
     WHILE ( ABS(E1-E0) > 0.0000001 ) and ( Ktr <= 20 ) DO
       BEGIN
         E0:= E1;
         E1:= E0 - ( ( E0 - e*SIN(E0)-m ) / ( 1.0 - e*COS(E0) ) );
         INC( Ktr );
       END;

     IF Ktr > 20 THEN
         WriteLn( 'NewtonRhapson not converged in 20 Iterations' );

     { ------------- Find True Anomaly at Epoch  ----------------- }
     Sinv:= ( SQRT( 1.0-e*e ) * SIN(E1) ) / ( 1.0-e*COS(E1) );
     Cosv:= ( COS(E1)-e ) / ( 1.0 - e*COS(E1) );
     NU  := ATAN2( Sinv,Cosv );
   END;  { Procedure NewtonR }
{
```

```
}
{ ----------------------------------------------------------------------------
|
|                             PROCEDURE KEPLER
|
|  This procedure solves Keplers problem for orbit determination and returns a
|    future Geocentric Equatorial (IJK) position and velocity vector.  The
|    solution procedure uses Universal variables.
|
|  Algorithm     : Initialize variables
|                  Find size and shape parameters for all cases
|                  Setup initial guesses with IF statements
|                  Loop while the time has not converged
|                  If too many iterations, print an error
|                     otherwise calculate the answer
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
|  Inputs        :
|    Ro          - IJK Position vector - initial          DU
|    Vo          - IJK Velocity vector - initial          DU / TU
|    Time        - Length of time to propagate            TU
|
|  OutPuts       :
|    R           - IJK Position vector                    DU
|    V           - IJK Velocity vector                    DU / TU
|
|  Locals        :
|    F           - f expression
|    G           - g expression
|    FDot        - f dot expression
|    GDot        - g dot expression
|    XOld        - Old Universal Variable X
|    XOldSqrd    - XOld squared
|    XNew        - New Universal Variable X
|    XNewSqrd    - XNew squared
|    ZNew        - New value of z
|    CNew        - C(z) function
|    SNew        - S(z) function
|    DeltaT      - change in t                            TU
|    TimeNew     - New time                               TU
|    RDotV       - Result of Ro dot Vo
|    A           - Semi major axis                        DU
|    Alpha       - Reciprocol  1/a
|    SME         - Specific Mech Energy                   DU2 / TU2
|    Period      - Time period for satellite              TU
|    S           - Variable for parabolic case
|    W           - Variable for parabolic case
|    Temp        - Temporary Extended value
|    i           - index
|
|  Constants     :
|    HalfPi
|    TwoPi
|    Small       - Tolerance for roundoff errors
|    Infinite    - Flag for an indefinite element
|
|  Coupling      :
|    MAG           Magnitude of a vector
|    DOT           Dot product of two vectors
|    REALMOD       REAL MOD function
|    COT           Cotangent function
|    POWER         Raise a number to some power
|    SGN           Sign of a number +1 or -1
|    FindCandS     Find C and S functions
|    Tan           Tangent of a value
|
|  References    :
|    Kaplan        pg. 304-308  ( Includes first guess for x if parabolic)
|    BMW           pg. 191-199, 203-212
|
 ---------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE Kepler                ( Ro,Vo              : Vector;
                                  Time               : Extended;
                                  VAR R,V            : Vector    );
   CONST
      HalfPi   : Extended = 1.57079632679490;
      TwoPi    : Extended = 6.28318530717959;
      Small    : Extended = 0.000001;
      Infinite : Extended = 999999.9;
   VAR
      F, G, FDot, GDot, DeltaT, XOld, XOldSqrd, XNew, XNewSqrd, ZNew,
      CNew, SNew, TimeNew, RDotV, A, Alpha, SME, Period, S, W, Temp : Extended;
      i : Integer;
   BEGIN
     { --------------------  Initialize values   ------------------- }
     TimeNew := -10.0;
     FOR i:= 1 to 4 DO
         V[i]:= 0.0;
     MAG( Ro );
     MAG( Vo );
     RDotV:= DOT( Ro,Vo );

     { -------------  Find SME, Alpha, and A  --------------------- }
     SME:= ( Vo[4]*Vo[4]/2.0 ) - ( 1.0/Ro[4] );
     Alpha:= -SME*2.0;

     IF ABS( SME ) > Small THEN  { circle, ellipse, hyperbola }
         A:= -1.0 / ( 2.0*SME )
       ELSE
         A:= Infinite;
     IF ABS( Alpha ) < Small THEN { Parabola }
         Alpha:= 0.0;

     { ---------------   Setup initial guess for x  --------------- }
             { ----------  Circle and Ellipse -------------- }
     IF Alpha >= Small THEN
       BEGIN
         Period:= TwoPi * SQRT( POWER( ABS(A),3.0 ) );
         IF ABS( Time ) > ABS( Period ) THEN
             Time:= RealMOD( Time,Period );
         IF ABS(Alpha-1.0) > Small THEN
              XOld := Time * Alpha
           ELSE
           { - 1st guess can't be too close. ie a circle, r=a - }
              XOld:= Time*Alpha*0.97;
       END
       ELSE
             { ----------------  Parabola  --------------- }
         IF ABS( Alpha ) < Small THEN
           BEGIN
             S:= 0.5 * (HalfPi - ARCTAN( 3.0*SQRT( 1.0/POWER(Ro[4],3.0) )* Time ) );
             W:= ARCTAN( POWER( TAN( S ) ,1.0/3.0 ) );
             XOld := SQRT(Ro[4])*( 2.0*COT(2.0*W) );
             Alpha:= 0.0;
           END
           ELSE
             { ---------------  Hyperbola  --------------- }
             BEGIN
               Temp:= -2.0*Time /
                      ( A*( RDotV + SGN(Time)*SQRT(-A)*(1.0-Ro[4]/a) ) );
               XOld:= SGN( Time ) * SQRT( -A ) * Ln( Temp );
             END;

{
```

```
}
    i:= 1;
    WHILE ( ABS( TimeNew-Time ) > 0.000001 ) and ( i <= 15 ) DO
      BEGIN
        XOldSqrd := XOld*XOld;
        ZNew     := XOldSqrd * Alpha;

    { ------------------ Find C and S functions ----------------- }
        FindCandS( ZNew, CNew,SNew );

    { ---------- Use a Newton iteration for new values  ---------- }
        TimeNew := XOldSqrd*XOld*SNew + RDotV*XOldSqrd*CNew +
                   Ro[4]*XOld*( 1.0 - ZNew*SNew );
        DeltaT  := XOldSqrd*CNew + RDotV*XOld*( 1.0 - ZNew*SNew ) +
                   Ro[4]*( 1.0 - ZNew*CNew );

    { ---------------  Calculate new value for x   -------------- }
        XNew := XOld + ( Time-TimeNew ) / DeltaT;
    { --------------------------------------------------------------
    | Check if the orbit is an ellipse and xnew > 2pi SQRT(a), the step
    |   size must be changed.  This is accomplished by multiplying DeltaT
    |   by 10.0.  NOTE !! 10.0 is arbitrary, but seems to produce good
    |   results.  The idea is to keep XNew from increasing too rapidily.
     ---------------------------------------------------------------- }
        IF ( A > 0.0 ) and ( ABS(XNew)>TwoPi*SQRT(A) ) and ( SME < 0.0 ) THEN
            XNew := XOld + ( Time-TimeNew ) / ( DeltaT*10.0 );
{}
{}      IF Show = 'Y' THEN
{}          Writeln( i:2,XOld:10:5,' ',TimeNew:10:5,' ',DeltaT:10:5,' ',
{}                   XNew:10:5,SNew:10:5,CNew:10:5,znew:10:5 );
{}      IF Show = 'S' THEN
{}          Writeln( FileOut,i:2,XOld:10:5,' ',TimeNew:10:5,' ',DeltaT:10:5,' ',
{}                   XNew:10:5,SNew:10:5,CNew:10:5,znew:10:5 );
{}
        Inc( i );
        XOld := XNew;
    END;  { While finding Universal Variables }

    IF i >= 15 THEN
        WriteLn( ' Not converged in 15 iterations ' )
      ELSE
      BEGIN

    { ---  Calculate position and velocity vectors at new time ---- }
        XNewSqrd := XNew*XNew;
        F := 1.0 - ( XNewSqrd*CNew / Ro[4] );
        G := Time - XNewSqrd*XNew*SNew;
        FOR i:= 1 to 3 DO
            R[i]:= F*Ro[i] + G*Vo[i];
        MAG( R );
        GDot := 1.0 - ( XNewSqrd*CNew / R[4] );
        FDot := ( XNew / ( Ro[4]*R[4] ) ) * ( ZNew*SNew - 1.0 );
        FOR i:= 1 to 3 DO
            V[i]:= FDot*Ro[i] + GDot*Vo[i];
        MAG( V );
      END;
  END;  { Procedure Kepler }
{
```

```
}
{ ---------------------------------------------------------------------
|
|                          PROCEDURE GAUSS
|
| This procedure solves the Gauss problem of orbit determination and returns
|   the velocity vectors at each of two given position vectors.  The solution
|   uses Universal Variables for calculation and a bissection technique for
|   updating Z.  This method is slower than the Newton iteration discussed in
|   BMW, but it does NOT suffer problems with negative z values, and is valid
|   for ellipses LESS THAN one revolution, parabolas, and Hyperbolas.  Also
|   note the selection of small since the algorithm is very sensitive to
|   changes in this variable.  A value of 0.001 will converge in say 10
|   iterations instead of 25 iterations for a value of 0.000001, and the
|   accuracy will differ in the 3rd-4th decimal place.  I chose to keep the
|   higher accuracy for cases like example 13, BMW pg. 274, #5.10.
|   ( Refer to graph on BMW pg. 235 for ranges of z. )
|
| Algorithm     : Initialize variables and setup initial guesses
|                 Loop while the time has not converged
|                 If too many iterations, print an error
|                    otherwise calculate the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| Inputs        :
|   R1          - IJK Position vector 1                  DU
|   R2          - IJK Position vector 2                  DU
|   DM          - direction of motion                    'L','S'
|   Time        - Time between R1 and R2                 TU
|
| OutPuts       :
|   V1          - IJK Velocity vector                    DU / TU
|   V2          - IJK Velocity vector                    DU / TU
|
| Locals        :
|   VarA        - Variable of the iteration, NOT the semi major axis!
|   Y           -
|   Upper       - Upper bound for Z
|   Lower       - Lower bound for Z
|   CosDeltaNu  - Cosine of true anomaly change          rad
|   F           - f expression
|   G           - g expression
|   GDot        - g dot expression
|   XOld        - Old Universal Variable X
|   XOldCubed   - XOld cubed
|   ZOld        - New value of z
|   ZNew        - New value of z
|   CNew        - C(z) function
|   SNew        - S(z) function
|   TimeNew     - New time                               TU
|   Small       - Tolerance for roundoff errors
|   i           - index
|   j           - index
|
| Constants     :
|   TwoPi
|   Small       - Tolerance for roundoff errors
|
| Coupling      :
|   MAG           Magnitude of a vector
|   DOT           Dot product of two vectors
|   FindCandS     Find C and S functions
|
| References    :
|   BMW           pg. 228-241  (Uses a Newton iteration)
|
 --------------------------------------------------------------------- }
{
```

```
}
PROCEDURE GAUSS                 ( R1,R2                    : Vector;
                                  DM                       : Char;
                                  Time                     : Extended;
                                  VAR V1,V2                : Vector    );
   CONST
      TwoPi     : Extended = 6.28318530717959;
      Small     : Extended = 0.000001;
   VAR
      VarA, Y, Upper, Lower, CosDeltaNu, F, G, GDot, XOld, XOldCubed,
      ZOld, ZNew, CNew, SNew, TimeNew  : Extended;
      i, j                             : Integer;
   BEGIN
     { -------------------- Initialize values   ------------------ }
     TimeNew:= -10.0;
     MAG(R1);
     MAG(R2);
     FOR i:= 1 to 4 DO
       BEGIN
         V1[i]:= 0.0;
         V2[i]:= 0.0;
       END;
     CosDeltaNu:= DOT(R1,R2)/(R1[4]*R2[4]);
     IF Dm = 'L' THEN
         VarA := -SQRT( R1[4]*R2[4]*(1.0+CosDeltaNu) )
       ELSE
         VarA :=  SQRT( R1[4]*R2[4]*(1.0+CosDeltaNu) );

     { --------------- Form Initial guesses   ------------------- }
     ZOld := 0.0;
     CNew := 0.5;
     SNew := 1.0/6.0;
     Upper:= TwoPi*TwoPi;  { Bounds for Z iteration }
     Lower:= -2.0*TwoPi;

     { -------- Determine if the orbit is possible at all --------- }
     IF ABS( VarA ) > Small THEN
       BEGIN

     { ----------------------------------------------------------
     |  Perform Gaussian Iteration using Universal Variables.  Notice
     |   the iteration is performed using a bissection technique instead of
     |   a Newton itertion.  Although the Newton iteration is quicker, the
     |   bissection will not fail with large negative Z values.  The upper
     |   and lower bounds are adjusted as required to keep y from becoming
     |   negative.
       ---------------------------------------------------------- }
{
```

```
}
        i:= 0;
        WHILE ( ABS( TimeNew-Time ) > Small ) and ( i <=30 ) DO
          BEGIN
            Y:= R1[4] + R2[4] - ( VarA*(1.0-ZOld*SNew)/SQRT(CNew) );

      { ------------------------------------------------------------
      | A check is needed for special cases where VarA is greater than 0.0.
      |  It's possible that Z can become very negative, and cause the square
      |  root in the XOld calculation to blow up.  This section loops and
      |  adjusts the upper and lower bounds until the ZNew value will
      |  result in a + y value.  The solution is to slowly update the lower
      |  bound of Z until y is +.  The 0.8* for ZNew is simply a means to let
      |  Z change a little slower.  The ZNew equation is found by solving the
      |  y equation for z when y = 0.
        ------------------------------------------------------------ }
            IF ( VarA > 0.0 ) and ( Y < 0.0 ) THEN
              BEGIN
                j:= 1;
                WHILE ( Y < 0.0 ) and ( j < 10 ) DO
                  BEGIN
                    ZNew:= 0.8*(1.0/SNew)*( 1.0 - (R1[4]+R2[4])*SQRT(CNew)/VarA  );

      { ------------------ Find C and S functions ------------------ }
                    FindCandS( ZNew, CNew,SNew );
                    ZOld:= ZNew;
                    Lower:= ZOld;
                    Y:= R1[4] + R2[4] - ( VarA*(1.0-ZOld*SNew)/SQRT(CNew) );
                    INC( j );
                  END;
                 IF j >= 10 THEN
                     WriteLn( 'The Iterations failed for Yn in GAUSS' );
              END;

            XOld      := SQRT( Y/CNew );
            XOldCubed:= XOld*XOld*XOld;
            TimeNew  := XOldCubed*SNew + VarA*SQRT(Y);

      { ---------------  Readjust upper and lower bounds  ----------- }
            IF TimeNew < Time THEN
                Lower:= ZOld;
            IF TimeNew > Time THEN
                Upper:= ZOld;

            ZNew:= ( Upper+Lower ) / 2.0;
{}
{}      IF Show = 'Y' THEN
{}          Writeln( i:2,ZOld:10:5,Y:10:5,XOld:10:5,TimeNew:10:5,VarA:7:3,upper:9:5,lower
{}      IF Show = 'S' THEN
{}          Writeln( FileOut,i:2,ZOld:10:5,Y:10:5,XOld:10:5,TimeNew:10:5,VarA:7:3,upper:9
{}
      { ------------------ Find C and S functions ------------------ }
            FindCandS( ZNew, CNew,SNew );
            ZOld := ZNew;
            Inc( i );

      { -------  Make sure the first guess isn't too close ---------- }
            IF ( ABS( TimeNew - Time ) < Small ) and ( i = 1 ) THEN
                TimeNew:= -10.0;
        END;  { While loop }

        IF i >= 30 THEN
            Write( 'Gauss not converged in 30 iterations ' )
          ELSE
          BEGIN

      { ------  Use F and G series to find Velocity Vectors  -------- }
            F := 1.0 - ( Y / R1[4] );
            G := VarA*SQRT( Y );
            GDot := 1.0 - Y/R2[4];
            FOR i:= 1 to 3 DO
              BEGIN
                V1[i]:= ( R2[i] - F*R1[i] )/G;
                V2[i]:= ( GDot*R2[i] - R1[i] )/G;
              END;
            MAG( V1 );
            MAG( V2 );
          END;  { If the answer has converged }
      END  { IF Var A > 0.0 }
      ELSE
        WriteLn( ' Gauss problem cannot be solved ' );

  END;  { Procedure Gauss }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                       PROCEDURE IJKtoLATLON
|
|  This procedure converts a Geocentric Equatorial (IJK) position vector into
|    latitude and longitude.  Geodetic and Geocentric latitude are found.
|
|  Algorithm     : Initialize variables
|                  Find the longitude being careful to reslove the angle
|                  Setup iteration for latitude
|                  Loop while the deltas are not equal
|                  Write an error message if the values do not converge
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  18 Sep 1990
|
|  Inputs        :
|    R           - IJK position vector                   DU
|    JD          - Julian Date                           days from 4713 B.C.
|
|  OutPuts       :
|    GeoCnLat    - Geocentric Latitude                   -Pi to Pi rad
|    Lon         - Longitude (WEST -)                    -2Pi to 2Pi rad
|
|  Locals        :
|    Rc          - Range of site w.r.t. earth center     DU
|    Height      - Height above earth w.r.t. site        DU
|    Alpha       - Angle from I axis to point, LST       rad
|    OldDelta    - Previous value of DeltaLat            rad
|    DeltaLat    - Diff between Delta and Geocentric lat rad
|    GeoDtLat    - Geodetic Latitude                     rad
|    TwoFMinusF2 - 2*F - F squared
|    OneMinusF2  - ( 1 - F ) squared
|    Delta       - Declination angle of R in IJK system  rad
|    RSqrd       - Magnitude of r squared                DU2
|    SinTemp     - Sine of Temp                          rad
|    Temp        - Diff between Geocentric/Geodetic lat  rad
|    GST         - Greenwich Siderual time               rad
|    i           - index
|
|  Constants     :
|    Pi
|    TwoPi
|    Flat        - Flatenning of the Earth
|    Small       - Tolerance
|
|  Coupling      :
|    MAG           Magnitude of a vector
|    Atan2         Arc Tangent which also resolves quadrant
|    Power         Raises a value to some power
|    ArcSin        Arc Sine of a value
|    GSTime        Greenwich Sidereal Time
|    RealMOD       Extended MOD function
|
|  References    :
|    Escobal       pg. 398-399
|
 ---------------------------------------------------------------------- }
{
```

```
}
PROCEDURE IJKtoLatLon          ( R                          : Vector;
                                 JD                         : Extended;
                                 VAR GeoCnLat,Lon           : Extended );
   CONST
      Pi       : Extended = 3.14159265358979;
      TwoPi    : Extended = 6.28318530717959;
      Small    : Extended = 0.000001;
      Flat     : Extended = 0.003352810664747352;
   VAR
      Rc, Height, Alpha, OldDelta, DeltaLat, GeoDtLat, TwoFMinusF2, RSqrd,
      OneMinusF2, Delta, SinTemp,  Temp, GST : Extended;
      i                                      : Integer;
   BEGIN
     { -------------------- Initialize values   ------------------- }
     MAG( R );
     TwoFMinusF2:= 2.0*Flat - Flat*Flat;
     OneMinusF2 := POWER( '.0-Flat,2.0 );

     { ----------------- Find Longitude value  --------------------- }
     Temp := SQRT( R[1]*R[1] + R[2]*R[2] );
     Alpha:= ATan2( R[2] / Temp , R[1] / Temp );
     GST  := GSTIME( JD );
     Lon  := Alpha - GST;
     IF ABS(Lon) >= Pi THEN
         IF Lon < 0.0 THEN
             Lon:= TwoPi + Lon
           ELSE
             Lon:= Lon - TwoPi;

     { --------------- Set up initial latitude value  -------------- }
     Delta := ArcTan( R[3] / Temp );
     IF ABS( Delta ) > Pi THEN
         Delta:= RealMOD( Delta,Pi );
     GeoCnLat:= Delta;
     OldDelta:=  1.0;
     DeltaLat:= 10.0;
     RSqrd := R[4]*R[4];

     { ----- Iterate to find Geocentric and Geodetic Latitude  ----- }
     i:= 1;
     WHILE ( ABS( OldDelta - DeltaLat ) > 0.00001 ) and ( i < 10 ) DO
       BEGIN
         OldDelta:= DeltaLat;
         Rc       := SQRT( ( 1.0-TwoFMinusF2 ) /
                           ( 1.0-TwoFMinusF2*COS(GeoCnLat)*COS(GeoCnLat) ) );
         GeoDtLat := ArcTan( TAN(GeoCnLat) / OneMinusF2 );
         Temp     := GeoDtLat-GeoCnLat;
         SinTemp  := SIN( Temp );
         Height   := SQRT( RSqrd - Rc*Rc*SinTemp*SinTemp ) - Rc*COS(Temp);
         DeltaLat := ARCSIN( Height*SinTemp / R[4] );
         GeoCnLat := Delta - DeltaLat;
         INC( i );
       END;  { While }

     IF i >= 10 THEN
         WriteLn( 'IJKtoLa'. 'n did NOT converge ' );
   END;  { Procedure IJKtoL..Lon }

{
```

```
}
{ ----------------------------------------------------------------------------
|
|                              PROCEDURE SUN
|
|  This procedure calculates the Geocentric Equatorial position vector for
|    the Sun given the Julian Date.  This is the low precision formula and
|    is valid for years from 1950 to 2050.  Accuaracy of apparent coordinates
|    is 0.01 degrees.  Notice many of the calculations are performed in
|    degrees, and are not changed until later.  This is due to the fact that
|    the Almanac uses degrees exclusively in their formulations.
|
|  Algorithm     : Calculate the several values needed to find the vector
|                  Be careful of quadrant checks
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  25 Aug 1988
|
|  Inputs        :
|    JD          - Julian Date                            days from 4713 B.C.
|
|  Outputs       :
|    RSun        - IJK Position vector of the Sun         AU
|    RtAsc       - Right Ascension                        rad
|    Decl        - Declination                            rad
|
|  Locals        :
|    MeanLong    - Mean Longitude
|    MeanAnomaly - Mean anomaly
|    N           - Number of days from 1 Jan 2000
|    EclpLong    - Ecliptic longitude
|    Obliquity   - Mean Obliquity of the Ecliptic
|
|  Constants     :
|    Pi          -
|    TwoPi       -
|    Rad         - Degrees per radian
|
|  Coupling      :
|    RealMOD       Extended MOD function
|    ArcSin        Arc Sine function
|
|  References           :
|    1987 Astronomical Almanac Pg. C24
|
 --------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE Sun                  ( JD                         : Extended;
                                 VAR RSun                   : Vector;
                                 VAR RtAsc,Decl             : Extended );
   CONST
      Pi    : Extended =  3.14159265358979;
      TwoPi : Extended =  6.28318530717959;
      Rad   : Extended = 57.29577951308230;
   VAR
      MeanLong, MeanAnomaly, EclpLong, Obliquity, N : Extended;
   BEGIN
     { -------------------- Initialize values   ------------------ }
     N:= ( JD - 2451545.0 );

     MeanLong:= 280.460 + 0.9856474*N;
     MeanLong:= RealMOD( MeanLong,360.0 ); {deg}

     MeanAnomaly:= 357.528 + 0.9856003*N;
     MeanAnomaly:= RealMOD( MeanAnomaly/Rad,TwoPi ); {rad}
     IF MeanAnomaly < 0.0 THEN
         MeanAnomaly:= TwoPi + MeanAnomaly;

     EclpLong:= MeanLong + 1.915*sin(MeanAnomaly) + 0.020*sin(2.0*MeanAnomaly);{deg}
     Obliquity:= 23.439 - 0.0000004*N; {deg}

     MeanLong := MeanLong/Rad;
     IF MeanLong < 0.0 THEN
         MeanLong:= TwoPi + MeanLong;
     EclpLong := EclpLong / Rad;
     Obliquity:= Obliquity / Rad;

     RtAsc:= ARCTAN( Cos(Obliquity)*Tan(EclpLong) );

     { ---- Check that RtAsc is in the same quadrant as EclpLong --- }
     IF EclpLong < 0.0 THEN
         EclpLong:= EclpLong + TwoPi;   { make sure it's in 0 to 2pi range }
     IF ABS( EclpLong-RtAsc ) > Pi/2.0 THEN
         RtAsc:= RtAsc + 0.5*Pi*ROUND( (EclpLong-RtAsc)/(0.5*Pi) );

     Decl := ARCSIN( Sin(Obliquity)*Sin(EclpLong) );

     { --------- Find magnitude of SUN vector, then components ----- }
     RSun[4]:= 1.00014 - 0.01671*Cos( MeanAnomaly )
                       - 0.00014*Cos( 2.0*MeanAnomaly );   { in AU's }
     RSun[1]:= RSun[4]*Cos( EclpLong );
     RSun[2]:= RSun[4]*Cos(Obliquity)*Sin(EclpLong);
     RSun[3]:= RSun[4]*Sin(Obliquity)*Sin(EclpLong);

   END;  { Procedure Sun }

{
```

```
}
{ ------------------------------------------------------------------------
|
|                           PROCEDURE MOON
|
| This procedure calculates the Geocentric Equatorial (IJK) position vector
|   for the moon given the Julian Date.  This is the low precision formula and
|   is valid for years between 1950 and 2050.  Notice many of the calculations
|   are performed in degrees.  This coincides with the development in the
|   Almanac.  A few equations were split in two to prevent software problems
|   with numeric coprocessors.  The errors seemed to be a stack overflow problem
|   since the equation is so long.  The program results are as follows:
|
|               Eclpitic Longitude  0.3   degrees
|               Eclpitic Latitude   0.2   degrees
|               Horiz Parallax      0.003 degrees
|               Distance from Earth 0.2   DUs
|               Right Ascension     0.3   degrees
|               Declination         0.2   degrees
|
| Algorithm     : Find the initial quantities
|                 Calculate direction cosines
|                 Find the position and velocity vector
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  25 Aug 1988
|
| Inputs        :
|   JD          - Julian Date                            days from 4713 B.C.
|
| Outputs       :
|   RMoon       - IJK Position vector of the Moon        DU
|   RtAsc       - Right Ascension                        rad
|   Decl        - Declination                            rad
|
| Locals        :
|   EclpLong    - Ecliptic Longitude
|   EclpLat     - Eclpitic Latitude
|   HzParal     - Horizontal Parallax
|   l           - Geocentric Direction Cosines
|   m           -         "      "
|   n           -         "      "
|   Tu          - Julian Centuries from 1 Jan 1900
|   x           - Temporary REAL variable
|
| Constants     :
|   TwoPi       -                                        6.28318530717959
|   Rad         - Degrees per radian                     57.29577951308230
|
| Coupling      :
|   RealMOD       Real MOD function
|   ArcSin        Arc Sine function
|   Atan2         Arc Tangent formula which resolves quadrants
|
| References    :
|   1987 Astronomical Almanac Pg. D46
|   Explanatory Supplement( 1960 ) pg. 106-111
|   Roy, Orbital Motion Pg. 61-62 ( Discussion of parallaxes )
|
 ----------------------------------------------------------------------- }
{
```

```
}
PROCEDURE Moon                    ( JD                          : Extended;
                                    VAR RMoon                   : Vector;
                                    VAR RtAsc,Decl              : Extended );
   CONST
      TwoPi  : Extended =  6.28318530717959;
      Rad    : Extended = 57.29577951308230;
   VAR
      EclpLong, EclpLat, HzParal, l,m,n,Tu,x : Extended;
   BEGIN
     { -------------------- Initialize values   ------------------- }
     Tu := ( JD - 2451545.0 ) / 36525.0;

     x  :=   218.32 + 481267.883*Tu
                  + 6.29*Sin( (134.9+477198.85*Tu)/Rad )
                  - 1.27*Sin( (259.2-413335.38*Tu)/Rad )
                  + 0.66*Sin( (235.7+890534.23*Tu)/Rad );

     EclpLong:= x + 0.21*Sin( (269.9+954397.70*Tu)/Rad )
                  - 0.19*Sin( (357.5+ 35999.05*Tu)/Rad )
                  - 0.11*Sin( (186.6+966404.05*Tu)/Rad );   { Deg }

     EclpLat :=     5.13*Sin( ( 93.3+483202.03*Tu)/Rad )
                  + 0.28*Sin( (228.2+960400.87*Tu)/Rad )
                  - 0.28*Sin( (318.3+  6003.18*Tu)/Rad )
                  - 0.17*Sin( (217.6-407332.20*Tu)/Rad );     { Deg }

     x :=    0.9508 + 0.0518*Cos( (134.9+477198.85*Tu)/Rad );

     HzParal := x + 0.0095*Cos( (259.2-413335.38*Tu)/Rad )
                  + 0.0078*Cos( (235.7+890534.23*Tu)/Rad )
                  + 0.0028*Cos( (269.9+954397.70*Tu)/Rad );   { Deg }

     EclpLong := RealMOD( EclpLong/Rad, TwoPi );
     EclpLat  := RealMOD( EclpLat/Rad, TwoPi );
     HzParal  := RealMOD( HzParal/Rad, TwoPi );

     { ------------- Find the geocentric direction cosines --------- }
     l:= COS( EclpLat ) * Cos( EclpLong );
     m:= 0.9175*Cos(EclpLat)*Sin(EclpLong) - 0.3978*Sin(EclpLat);
     n:= 0.3978*Cos(EclpLat)*Sin(EclpLong) + 0.9175*Sin(EclpLat);

     { -------------- Calculate Moon position vector -------------- }
     RMoon[4]:= 1.0/SIN( HzParal );
     RMoon[1]:= RMoon[4]*l;
     RMoon[2]:= RMoon[4]*m;
     RMoon[3]:= RMoon[4]*n;

     { --------------- Find Rt Ascension and Declination ----------- }
     RtAsc:= ATan2( m,l );
     Decl:= ArcSin( n );

   END;  { Procedure Moon }

{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE PLANETRV
|
|  This procedure calculate the planetary ephemerides using the Epoch J2000.
|    The coefficients are obtained from Danbys book and provisions are left
|    to obtain Heliocentric Equatorial, or Heliocentric Ecliptic coordinates.
|    Notice the ephemeris presents data wrt the solar equator.
|
|  Algorithm     : Use a case statement to assign each planets values
|                  Find the vectors
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109    4 Feb 1990
|
|  Inputs        :
|    PlanetNum   - Number of planet                         1..9
|    JD          - Julian Date                              days from 4713 B.C.
|
|  OutPuts       :
|    R           - XYZ position vector                      AU
|    V           - XYZ velocity vector                      AU / TU
|
|  Locals        :
|    u           -
|    l           -
|    cappi       -
|    TU          -
|    N           -
|    obliquity   -
|    a           -
|    e           -
|    p           -
|    inc         -
|    omega       -
|    argp        -
|    nu          -
|    m           -
|    LLong       -
|    LongP       -
|    e0          -
|
|  Constants     :
|    TwoPi       -
|    Rad         - Degrees per radian
|
|  Coupling      :
|    RealMOD
|    NewtonR
|    RandV
|
|  References    :
|    Danby          pg. 427-429
|    Escobal(2)     pg. 261-270
|    Astronomical Almanac pg E2-E4
|
 ----------------------------------------------------------------------- }

PROCEDURE PlanetRV              ( PlanetNum               : Integer;
                                  JD                      : Extended;
                                  VAR R,V                 : Vector   );
   CONST
      TwoPi : Extended =  6.28318530717959;
      Rad   : Extended = 57.29577951308230;
   VAR
      TUDaySun,u,l,cappi,Tu,n,obliquity,a,e,p,inc,omega,argp,nu,
      llong,longp,m,e0        : Extended;
      i          : Integer;
   BEGIN
     Tu := ( JD - 2451545.0 ) / 36525.0;
{
```

```
}
CASE PlanetNum OF
  1: BEGIN  { -----------Mercury ---------}
       LongP:=     1.3518643 +   0.0271656*Tu + 0.000005166*Tu*Tu;
       Omega:=     0.8435332 +   0.0207029*Tu + 0.000003072*Tu*Tu;
       Inc  :=     0.1222601 +   0.0000318*Tu - 0.000000314*Tu*Tu;
       e    :=     0.2056318 +   0.0000204*Tu - 0.000000030*Tu*Tu;
       LLong:=     4.4026098 +2608.8147071*Tu + 0.000005306*Tu*Tu;
       a    :=     0.3871035;
     END;
  2: BEGIN  { -----------Venus  ---------}
       LongP:=     2.2962199 +   0.0244734*Tu - 0.000018727*Tu*Tu
                                              - 0.000000087*Tu*Tu*Tu;
       Omega:=     1.3383171 +   0.0157275*Tu + 0.000007103*Tu*Tu;
       Inc  :=     0.0592480 +   0.0000175*Tu - 0.000000017*Tu*Tu;
       e    :=     0.0067719 -   0.0000478*Tu;
       LLong:=     3.1761467 +1021.3529430*Tu + 0.000005428*Tu*Tu;
       a    :=     0.7233074;
     END;
  3: BEGIN  { -----------Earth  ---------}
       LongP:=     1.7965956 +   0.0300116*Tu + 0.000008029*Tu*Tu;
       Omega:=     0.0000000;
       Inc  :=     0.0000000;
       e    :=     0.0167086 -   0.0000420*Tu;
       LLong:=    17.4614336 + 628.3319667*Tu + 0.000005306*Tu*Tu;
       a    :=     1.0000116;
     END;
  4: BEGIN  { -----------Mars   ---------}
       LongP:=     5.8653576 +   0.0321323*Tu + 0.000000236*Tu*Tu;
       Omega:=     0.8649519 +   0.0134756*Tu + 0.000000279*Tu*Tu;
       Inc  :=     0.0322838 -   0.0000105*Tu + 0.000000227*Tu*Tu;
       e    :=     0.0934006 +   0.0000905*Tu - 0.000000080*Tu*Tu;
       LLong:=     6.2034809 + 334.0856279*Tu + 0.000005428*Tu*Tu;
       a    :=     1.5237107;
     END;
  5: BEGIN  { -----------Jupiter ---------}
       LongP:=     6.5333138 +   0.0281458*Tu + 0.000017994*Tu*Tu
                                              - 0.000000070*Tu*Tu*Tu;
       Omega:=     1.7534353 +   0.0178190*Tu + 0.000006999*Tu*Tu;
       Inc  :=     0.0227464 -   0.0000959*Tu + 0.000000087*Tu*Tu;
       e    :=     0.0484949 +   0.0001632*Tu - 0.000000470*Tu*Tu;
       LLong:=     0.5995465 +  52.9934808*Tu + 0.000003910*Tu*Tu;
       a    :=     5.2102156;
     END;
  6: BEGIN  { -----------Saturn  ---------}
       LongP:=     1.6241473 +   0.0342741*Tu + 0.000014626*Tu*Tu
                                              + 0.000000087*Tu*Tu*Tu;
       Omega:=     1.9838376 +   0.0153082*Tu - 0.000002112*Tu*Tu
                                              - 0.000000035*Tu*Tu*Tu;
       Inc  :=     0.0434391 -   0.0000652*Tu - 0.000000262*Tu*Tu;
       e    :=     0.0555086 -   0.0003468*Tu - 0.000001000*Tu*Tu;
       LLong:=     0.8740168 +  21.3542956*Tu + 0.000009076*Tu*Tu;
       a    :=     9.5380701;
     END;
  7: BEGIN  { -----------Uranus  ---------}
       LongP:=     3.0195096 +   0.0259422*Tu + 0.000003752*Tu*Tu;
       Omega:=     1.2916474 +   0.0090954*Tu + 0.000023387*Tu*Tu
                                              + 0.000000332*Tu*Tu*Tu;
       Inc  :=     0.0134948 +   0.0000135*Tu + 0.000000646*Tu*Tu;
       e    :=     0.0462959 -   0.0000273*Tu + 0.000000080*Tu*Tu;
       LLong:=     5.4812939 +   7.5025431*Tu + 0.000005306*Tu*Tu;
       a    :=    19.1833020;
     END;
  8: BEGIN  { -----------Neptune ---------}
       LongP:=     0.8399169 +   0.0248931*Tu + 0.000006615*Tu*Tu;
       Omega:=     2.3000657 +   0.0192371*Tu + 0.000004538*Tu*Tu;
       Inc  :=     0.0308915 -   0.0001625*Tu - 0.000000140*Tu*Tu;
       e    :=     0.0089881 +   0.0000064*Tu;
       LLong:=     5.3118863 +   3.8376877*Tu + 0.000005393*Tu*Tu;
       a    :=    30.0551440;
     END;
  9: BEGIN  { -----------Pluto  ---------}
       LongP:=     3.9202678;
       Omega:=     1.9269569;
       Inc  :=     0.2990156;
       e    :=     0.2508770;
       LLong:=     3.8203049;
       a    :=    39.5375800;
     END;
END;  { Case }
{
```

```
}
    LLong:= REALMOD( LLong,TwoPI );
    LongP:= REALMOD( LongP,TwoPI );
    Omega:= REALMOD( Omega,TwoPI );

    Argp:= LongP - Omega;
    M := LLong - LongP;

    NewTonR( e,M,  E0,Nu );
    p:= a*(1.0-e*e);

    u := 0.0;
    l := 0.0;
    Cappi:= 0.0;

    RANDV( P,e,Inc,Omega,Argp,Nu,U,L,CapPi, R,V );
{
|  Alternate method for finding position vector
|    r[4]:= ( a*( 1.0-e*e) ) / ( 1.0+e*Cos(Nu) );
|    r[1]:= r[4]*( Cos(Nu+Argp)*Cos(Omega)-Sin(Nu+Argp)*Cos(Inc)*Sin(Omega) );
|    r[2]:= r[4]*( Cos(Nu+Argp)*Sin(Omega)+Sin(Nu+Argp)*Cos(Inc)*Cos(Omega) );
|    r[3]:= r[4]*Sin(Nu+Argp)*Sin(Inc);
}
    { ----- Calculations required for reference to mean equator --- }
    N := ( JD - 2451545.0 );
    Obliquity:= 23.439 - 0.0000004*N; {deg}
    Obliquity:= Obliquity / Rad;

    ROT1( R ,-Obliquity, R );
    ROT1( V ,-Obliquity, V );

    TUDaySun:= 54.20765355;  { days per sun TU}
    FOR i:= 1 to 3 DO
        v[i]:= v[i]/tudaysun;

    IF Show = 'Y' THEN
      BEGIN
        WriteLn( '      a            e            i         Omega       LongP  ');
        WriteLn( a:10:6,E:13:6,Inc*rad:10:6,Omega*rad:12:6,LongP*rad:14:8 );
        WriteLn( '    LLong          Argp          M            Nu ');
        WriteLn( LLong*rad:14:5,Argp*rad:12:6,M*rad:12:6,Nu*rad:12:6 );
      END;
  END;  { Procedure PlanetRV }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                             FUNCTION GEOCENTRIC
|
| This Function converts from Geodetic to Geocentric latitude.  Notice that
|   (1-f) squared = 1-eSqrd.
|
| Algorithm     : Find the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| Inputs        :
|   Lat         - Geodetic Latitude                       -Pi/2 - Pi/2 rad
|
| Outputs       :
|   Geocentric  - Geocentric Latitude                     -Pi/2 - Pi/2 rad
|
| Locals        :
|   None.
|
| Constants     :
|   EESqrd      - Eccentricity of Earth squared           0.00669437999013
|
| Coupling      :
|   None.
|
| References    :
|   Escobal pg. 136
|   Kaplan pg. 332-336
|
 --------------------------------------------------------------------------- }

FUNCTION Geocentric            ( Lat                   : Extended ):Extended;
   CONST
      EESqrd : Extended = 0.00669437999013;
   BEGIN
     Geocentric:= ARCTAN( (1.0 - EESqrd)*TAN(Lat) );
   END; { Function Geocentric }

{ ---------------------------------------------------------------------------
|
|                             FUNCTION INVGEOCENTRIC
|
| This Function converts from Geocentric to Geodetic latitude.  Notice that
|   (1-f) squared = 1-eSquared.
|
| Algorithm     : Find the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
| Inputs        :
|   Lat         - Geocentric Latitude                     -Pi/2 - Pi/2 rad
|
| Outputs       :
|   InvGeocentric- Geodetic Latitude                       -Pi/2 - Pi/2 rad
|
| Locals        :
|   None.
|
| Constants     :
|   EESqrd      - Eccentricity of Earth squared           0.00669437999013
|
| Coupling      :
|   None.
|
| References    :
|   Escobal pg. 136
|   Kaplan pg. 332-336
|
 --------------------------------------------------------------------------- }

FUNCTION InvGeocentric         ( Lat                   : Extended ):Extended;
   CONST
      EESqrd : Extended = 0.00669437999013;
   BEGIN
     InvGeocentric:= ARCTAN( TAN(Lat)/(1.0 - EESqrd) );
   END; { Function InvGeocentric }
```

```
}
{ ------------------------------------------------------------------------
|
|                        PROCEDURE SIGHT
|
| This procedure takes the position vectors of two satellites and determines
|   if there is line-of-sight between the two satellites.  A spherical Earth
|   with radius of 1 DU is assumed.  The process is to form the equation of
|   a line between the two vectors.  Differentiating and setting to zero finds
|   the minimum value, and when plugged back into the original line equation,
|   gives the minimum distance.  The parameter tmin is allowed to range from
|   0.0 to 1.0.
|
| Algorithm      : Find tmin
|                  Check value of tmin for LOS
|                  Find dist squared if needed
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109  31 Jan 1990
|
| Inputs         :
|   R1           - Position vector of the first sat       DU
|   R2           - Position vector of the second sat      DU
|
| Outputs        :
|   LOS          - Line of Sight                          'Yes','No '
|
| Locals         :
|   ADotB        - Dot product of a dot b
|   TMin         - Minimum value of t from a to b
|   DistSqrd     - Min Distance squared for to earth      DU
|   ASqrd        - Magnitude of A squared
|   BSqrd        - Magnitude of B squared
|
| Constants      :
|   None.
|
| Coupling:
|   DOT            Dot product of two vectors
|
| References     :
|   None.
|
 ------------------------------------------------------------------------ }

PROCEDURE SIGHT                ( R1,R2                    : Vector;
                                 VAR LOS                  : Str3      );
   VAR
     ADotB,TMin,DistSqrd,ASqrd,BSqrd: EXTENDED;
   BEGIN
     BSqrd:= R2[4]*R2[4];
     ASqrd:= R1[4]*R1[4];
     ADotB:= DOT( R1,R2 );
     TMin := ( ASqrd - ADotB ) / ( ASqrd + BSqrd - 2.0*ADotB );

     IF (TMin < 0.0) or (TMin > 1.0) THEN
         LOS:= 'YES'
       ELSE
       BEGIN
         DistSqrd:= (1.0-TMin)*ASqrd + ADotB*TMin;
         IF DistSqrd > 1.0 THEN
             LOS:= 'YES'
           ELSE
             LOS:= 'NO ';
       END;
   END;  { Procedure Sight }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                         PROCEDURE LIGHT
|
| This procedure determines if a spacecraft is sunlit or in the dark at a
|   particular time.  A spherical Earth and cylindrical shadow is assumed.
|
| Algorithm      : Find the sun vector
|                  Use the sight algorithm for the answer
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   9 Feb 1990
|
| Inputs         :
|   R            - IJK Position vector of satellite      DU
|   JD           - Julian Date of desired observation    Days from 4713 B.C.
|
| OutPuts        :
|   Vis          - Visibility Flag                       'Yes','No '
|
| Locals         :
|   RtAsc        - Suns Right ascension                  rad
|   Decl         - Suns Declination                      rad
|   RSun         - Sun vector                            AU
|   AUDU         - Conversion from AU to DU
|
| Constants      :
|   None
|
| Coupling       :
|   SUN            Position vector of Sun
|   LNCOM1         Multiple a vector by a constant
|   SIGHT          Does Line-of-sight exist bewteen vectors
|
| References     :
|   Escobal        pg.
|
 --------------------------------------------------------------------------- }

PROCEDURE LIGHT                ( R                       : Vector;
                                 JD                      : Extended;
                                 VAR LIT                 : Str3       );
   VAR
     RSun            : Vector;
     AUDU,RtAsc,Decl : Extended;
   BEGIN
     AUDU:= 149599650.0/6378.137;

     SUN( JD,RSun,RtAsc,Decl );
     LNCOM1( AUDU,RSun, RSun );

     { ---------- Is the satellite in the shadow? --------------- }
     SIGHT( RSun,R,  Lit );

   END; { Procedure Light }
{
```

```
}
{ ------------------------------------------------------------------------------
|
|                              PROCEDURE OMS2
|
|  This procedure determines the velocity and position vector of the shuttle
|    after it performs the OMS-2 burn.  Assume the burn and the resulting
|    velocity change are instanteanous.
|
|  Algorithm      : Find the velocity vector
|                   Rotate to IJK
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   7 Mar 1990
|
|  Inputs         :
|    Lat          - Geodetic latitude of the shuttle's Earth sub-
|                        point (its NADAR) before the burn.          rad
|    Lon          - Geodetic longitude of the shuttle's NADAR        rad
|    Alt          - Altitude of the shuttle above the Earth's surface  DU
|    Phi          - Shuttle flight path angle                        rad
|    Az           - Shuttle azimuth angle                            rad
|    Speed        - Shuttle scalar velocity with respect to inertial space DU/TU
|    JD           - Julian Date                                    Ref 4713 B.C.
|
|  Outputs :
|    R            - Position vector of the shuttle after the OMS2 burn DU
|    V            - Inertial velocity vector of the shuttle after OMS2 burn DU/TU
|
|  Locals         :
|    VSEZ         - Velocity vector expressed in the SEZ frame       DU/TU
|
|  Constants      :
|    HalfPi
|
|  Coupling       :
|    LSTIME       - Find LST and GST
|    SITE         - Find Site vector on an oblate Earth
|    ROT2         - Rotate about the 2 axis
|    ROT3         - Rotate about the 3 axis
|
|  References     :
|    None.
|
 ---------------------------------------------------------------------------- }

PROCEDURE OMS2                     ( Lat,Lon,Alt,Phi,Az,Speed,JD : Extended;
                                     VAR R,V                     : VECTOR  );
   CONST
     HalfPi  : Extended = 1.57079632679490;
   VAR
     GST, LST          : Extended;
     VSEZ,VS,TempVec : Vector;
   BEGIN
     LSTime( Lon,JD, Lst,Gst );
     SITE( Lat,Alt,Lst, R,VS );

      { -- Velocity vector in the rotating, Earth-fixed SEZ frame -- }
     VSEZ[1] := -Speed * COS(Phi) * COS(Az);
     VSEZ[2] :=  Speed * COS(Phi) * SIN(Az);
     VSEZ[3] :=  Speed * SIN(Phi);
     MAG( VSEZ );

      { ------------ Perform SEZ to IJK transformation ----------- }
     ROT2( VSEZ,   Lat-HalfPi, TempVec );
     ROT3( TempVec,   -LST    , V       );

   END; { Procedure OMS2 }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                              PROCEDURE RNGAZ
|
|  This procedure calculates the Range and Azimuth between two specified
|    ground points on a spherical Earth.  Notice the range will ALWAYS be
|    within the range of values listed since you do not know the direction of
|    firing, long or short.  The procedure will calculate Rotating Earth ranges
|    if the TOF is passed in other than 0.0.
|
|  Algorithm     : Find the range
|                  Calculate the Az noting all combinations of quadrants
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  25 Aug 1988
|
|  Inputs        :
|    LLat        - Start Geocentric Latitude             -Pi/2 -  Pi/2 rad
|    LLon        - Start Longitude (WEST -)              0.0 - 2Pi rad
|    TLat        - End Geocentric Latitude               -Pi/2 -  Pi/2 rad
|    TLon        - End Longitude (WEST -)                0.0 - 2Pi rad
|    TOF         - Time of Flight if ICBM, or 0.0        TU
|
|  OutPuts       :
|    Range       - Range between points                  0.0 -  Pi rad
|    Az          - Azimuth                               0.0 - 2Pi rad
|
|  Locals        :
|    None.
|
|  Constants     :
|    TwoPi
|    Pi
|    OmegaEarth  - Angular rotation of Earth                  Rad/TU
|    Small       - Tolerance
|
|  Coupling      :
|    ArcCos        Arc Cosine function
|
|  References    :
|    BMW           pg. 309-311
|
 -------------------------------------------------------------------------- }

PROCEDURE RngAz              ( LLat,LLon,TLat,TLon,TOF    : Extended;
                               VAR Range, Az              : Extended );
   CONST
      OmegaEarth  : Extended = 0.0588335906868878;
      Pi          : Extended = 3.14159265358979;
      TwoPi       : Extended = 6.28318530717959;
      Small       : Extended = 0.000001;
   BEGIN
     Range:= ArcCos( SIN(LLat)*SIN(TLat) +
                     COS(LLat)*COS(TLat)*COS(TLon-LLon + OmegaEarth*TOF) );

     { ------- Check if the Range is 0 or half the earth  ---------- }
     IF ABS( Sin(Range)*Cos(LLat) ) < Small THEN
         IF ABS( Range - Pi ) < Small THEN
             Az:= Pi
           ELSE
             Az:= 0.0
       ELSE
         Az:= ArcCos( ( SIN(TLat) - COS(Range) * SIN(LLat)) /
                      ( SIN(Range) * COS(LLat)) );

     { ------- Check if the Azimuth is grt than Pi ( 180deg ) ------ }
     IF SIN( TLon - LLon + OmegaEarth*TOF ) < 0.0 THEN
         Az:= TwoPi - Az;
   END;   { Procedure RngAz }

{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE PATH
|
|  This procedure determines the end position for a given range and azimuth
|    from a given point.  Notice the use of ATAN2 to eliminate quadrant
|    problems.  Also, Geocentric coordinates are used since the Earth is
|    assumed to be spherical.
|
|  Algorithm     : Find the latitude
|                  Find the change in longitude noting quadrant possibilities
|                  Calculate the longitude
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  25 Aug 1988
|
|  Inputs        :
|    LLat        - Start Geocentric Latitude              -Pi/2 - Pi/2 rad
|    LLon        - Start Longitude                        0.0 - 2Pi rad
|    Range       - Range between points                   DU
|    Az          - Azimuth                                0.0 - 2Pi rad
|
|  OutPuts       :
|    TLat        - End Geocentric Latitude                -Pi/2 - Pi/2 rad
|    TLon        - End Longitude                          0.0 - 2Pi rad
|
|  Locals        :
|    SinDeltaN   - Sine of Delta N                        rad
|    CosDeltaN   - Cosine of Delta N                      rad
|    DeltaN      - Angle bteween the two points           rad
|
|  Constants     :
|    Pi          -
|    TwoPi       -
|    Small       - Tolerance
|
|  Coupling      :
|    ArcSin        Arcsine function
|    RealMOD       Real MOD function
|    Atan2         Arc Tangent function which also resolves quadrants
|
|  References    :
|    BMW pg. 309-311
|
 ---------------------------------------------------------------------- }
{
```

```
}
PROCEDURE Path                  ( LLat, LLon, Range, Az       : Extended;
                                  VAR TLat, TLon              : Extended );
   CONST
      Pi    : Extended = 3.14159265358979;
      TwoPi: Extended = 6.28318530717959;
      Small: Extended = 0.000001;
   VAR
      SinDeltaN,CosDeltaN, DeltaN : Extended;
   BEGIN
     Az:= RealMOD( Az,TwoPi );
     IF LLon < 0.0 THEN
         LLon:= TwoPi + LLon;
     IF Range > TwoPi THEN
         Range:= RealMOD( Range,TwoPi );

     { ------------------ Find Geocentric Latitude ----------------- }
     TLat := ARCSIN( SIN(LLat)*COS(Range) + COS(LLat)*SIN(Range)*COS(Az) );

     { ----- Find Delta N, the angle between the points ------------ }
     IF (ABS(COS(TLat)) > Small) and (ABS(COS(LLat)) > Small) THEN
       BEGIN
         SinDeltaN:= SIN(Az)*SIN(Range) / COS(TLat);
         CosDeltaN:= ( COS(Range)-SIN(TLat)*SIN(LLat) ) / ( COS(TLat)*COS(LLat) );
         DeltaN:= ATan2(SinDeltaN,CosDeltaN);
       END
       ELSE
       BEGIN
     { ------- Case where launch is within 3nm of a Pole  ---------- }
         IF ABS(COS(LLat)) <= Small THEN
             IF (Range > Pi) and (Range < TwoPi) THEN
                 DeltaN:= Az + Pi
               ELSE
                 DeltaN:= Az;
     { ------- Case where end point is within 3nm of a pole -------- }
         IF ABS( COS(TLat) ) <= Small THEN
             DeltaN:= 0.0;
       END;

     TLon:= LLon + DeltaN;
     IF TLon < 0.0 THEN
         TLon:= TwoPi + TLon;
     IF TLon > TwoPi THEN
         TLon:= RealMOD( TLon,TwoPi );
   END;  { Procedure Path }
{
```

```
}
{ -----------------------------------------------------------------------------
|
|                         PROCEDURE TRAJEC
|
| This procedure calculates the Range, Azimuth, and Time of Flight between
|   two specified ground points for an ICBM with as known Q.  Calculations
|   depend on knowledge of burnout conditions, and the iterations are
|   performed for either a high or low trajectory.  Notice the ICBM will fly
|   on an inertial trajectory, and values for earth relative velocities,
|   etc., are calculated after the iteration.  Notice these calculations do
|   not support trajectories over half the world away.
|
| Algorithm     : Find the Range and Az with 0 TOF
|                   If the trajectory is possible,
|                     Loop to find the Range and Az corrected
|                     Calculate influence coefficients
|                     Find velocity needed
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   9 Oct 1988
|
| Inputs        :
|   LLat        - Start Geocentric Latitude              -Pi/2 - Pi/2 rad
|   LLon        - Start Longitude (WEST -)               0.0 - 2Pi rad
|   TLat        - End Geocentric Latitude                -Pi/2 - Pi/2 rad
|   TLon        - End Longitude (WEST -)                 0.0 - 2Pi rad
|   Rbo         - Radius at burnout                      DU
|   Q           - Non-dimensional Q performance based on Inertial Velocity
|   TypePhi     - Type of trajectory, High ot Low        'H', 'L'
|
| OutPuts       :
|   Range       - Rotating Range between points          0.0 -  Pi rad
|   Phi         - Inertial Flight Path Angle             rad
|   TOF         - Rotating Earth Time of Fligth          TU
|   Az          - Inert Azimuth                          0.0 - 2Pi rad
|   ICPhi       - Influence Coefficient for Phi          rad/rad
|   ICVbo       - Influence Coefficient for Vbo          rad/ DU/TU
|   ICRbo       - Influence Coefficient for Rbo          rad/rad
|   Vn          - Velocity the missile needs             DU/TU
|
| Locals        :
|   Small       - Tolerance
|   QBoMin      - Minimum Q for a given range
|   a           - Semi Major Axis                        DU
|   Ecc         - Eccentricity
|   E           - Eccentric Anomaly                      rad
|   RangeOld    - Iteration value of range               DU
|   Vbo         - Inertial Velocity                      DU/TU
|   VEarth      - Earths velocity                        DU/TU
|   i           - Index
|
| Constants     :
|   Pi          -
|   Rad         - Degrees per radian
|   OmegaEarth  - Angular rotation of Earth (Rad/TU)
|   Undefined   - Flag for an undefined element
|
| Coupling      :
|   MAG           Magnitude of a vector
|   ArcSin        Arc Sine of a value
|   ArcCos        Arc Cosine of a value
|   RngAz         Finds range and Azimuth given two points
|
| References    :
|   BMW           pg. 293-313
|
 ---------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE Trajec              ( LLat,LLon,TLat,TLon,Rbo,Q : Extended;
                                TypePhi                   : Char;
                                VAR Range,Phi,TOF,Az,ICPhi,
                                    ICVbo,ICRbo           : Extended;
                                VAR Vn                    : Vector    );
   CONST
      Pi          : Extended =  3.14159265358979;
      Small       : Extended =  0.000001;
      OmegaEarth  : Extended =  0.0588335906868878;
      Rad         : Extended = 57.2957795130823;
      Undefined   : Extended = 999999.1;
   VAR
      a,Ecc,E,RangeOld,Vbo,VEarth,QboMin : Extended;
      i                     : INTEGER;
   BEGIN
     { -----------------------   Initialize   ----------------------- }
     RangeOld := -1.0;
     i :=  1;

     { -------------- Iterate to find the flight time -------------- }
     RngAz( LLat,LLon,TLat,TLon,0.0, Range,Az );
     A := RBo / (2.0 - Q);

     QBoMin:= ( 2.0*Sin(Range/2.0) ) / (1.0+Sin(Range/2.0));

     IF (Q - QboMin) > 0.001 THEN
       BEGIN
         WHILE (ABS(RangeOld - Range) > Small) and ( i < 20 ) DO
           BEGIN
     { ----------- Check for High or Low Flight Path Angle --------- }
             IF TypePhi = 'H' THEN
                 Phi:= 0.5*( (Pi - ArcSIN(((2.0-Q)/Q)*Sin(Range/2.0)) )
                             - Range/2.0)
               ELSE
                 Phi:= 0.5*( ArcSIN(((2.0-Q)/Q)*Sin(Range/2.0)) - Range/2.0);
             Ecc := SQRT( 1.0 + Q*(Q-2.0)*Cos(Phi)*Cos(Phi) );
             E   := ArcCos( (Ecc-Cos(Range/2.0)) / (1.0-Ecc*Cos(Range/2.0)) );
             TOF := SQRT(A*A*A)*2.0*( Pi - E + Ecc*SIN(E) );

             IF Show = 'Y' THEN
{}               Writeln( i:4,Range*Rad:12:6,Phi*Rad:12:6,e*Rad:12:6,ecc:12:6,
{}                        TOF*13.44685108:12:6 );

             RangeOld:= Range;
             RngAz( LLat,LLon,TLat,TLon,TOF, Range,Az );
             i:= i+1;
           END;
         IF i >= 20 THEN
             WriteLn( 'The iteration did not converge in 20 steps' );

     { ----- Evaluate Influence Coefficients for unit errors ------- }
         VBo := SQRT( Q/Rbo );
         ICPhi:= ( ( 2.0*Sin(Range + 2.0*Phi) ) / Sin(2.0*Phi) ) - 2.0;
         ICVbo:= ( 8.0*Sin(Range/2.0)*Sin(Range/2.0) ) /
                 ( Vbo*Vbo*VBo*Rbo*Sin(2.0*Phi) );
         ICRbo:= ( 4.0*Sin(Range/2.0)*Sin(Range/2.0) ) /
                 ( Vbo*Vbo*RBo*Rbo*Sin(2.0*Phi) );

     { ------------ Find Velocity Needed, Relative Velocity -------- }
         VEarth:= OmegaEarth * Cos(LLat);

         VN[1]:= -VBo*COS( Phi )*COS(Az);
         VN[2]:=  VBo*COS( Phi )*SIN(Az) - VEarth;
         VN[3]:=  VBo*SIN( Phi );
         MAG( VN );
       END
       ELSE
       BEGIN
{}       Write( 'ICBM does not have enough energy  - ');
{}       Writeln( '  Q Min =',QboMin:12:6 );
         Phi  := Undefined;      TOF := Undefined;
         ICPhi:= Undefined;      ICVbo:= Undefined;
         ICRbo:= Undefined;      Vn[4]:= UNdefined;
       END;

   END;  { Procedure Trajec }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE HOHMANN
|
| This procedure calculates the delta v's for a Hohmann transfer for either
|   circle to circle, or ellipse to ellipse.  The notation used is from the
|   initial orbit (1) at point a, transfer is made to the transfer orbit (2),
|   and to the final orbit (3) at point b.
|
| Algorithm     : Find initial values
|                 If the orbits are both cir or ellip, find the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  19 Jun 1989
|
| Inputs        :
|   R1          - Initial position magnitude            DU
|   R3          - Final position magnitude              DU
|   e1          - Eccentricity of first orbit
|   e3          - Eccentricity of final orbit
|   Nu1         - True Anomaly of first orbit           0 or Pi rad
|   Nu3         - True Anomaly of final orbit           0 or Pi rad
|
| OutPuts       :
|   DeltaVa     - Change in velocity at point a         DU / TU
|   DeltaVb     - Change in velocity at point b         DU / TU
|   TOF         - Time of Flight for the transfer       TU
|
| Locals        :
|   SME1        - Specfic Mechanical Energy of first orbit   DU2 / TU
|   SME2        - Specfic Mechanical Energy of transfer orbitDU2 / TU
|   SME3        - Specfic Mechanical Energy of final orbit   DU2 / TU
|   V1          - Velocity of 1st orbit at point a           DU / TU
|   V2a         - Velocity of transfer orbit at point a      DU / TU
|   V2b         - Velocity of transfer orbit at point b      DU / TU
|   V3          - Velocity of final orbit at point b         DU / TU
|   a1          - Semi Major Axis of first orbit             DU
|   a2          - Semi Major Axis of Transfer orbit          DU
|   a3          - Semi Major Axis of final orbit             DU
|
| Constants     :
|   Pi
|
| Coupling      :
|   None.
|
| References    :
|   BMW           pg. 163-166
|
 ------------------------------------------------------------------------ }
{
```

```
}
PROCEDURE Hohmann              ( R1,R3,e1,e3,Nu1,Nu3      : Extended;
                                 VAR Deltava,Deltavb,TOF  : Extended );
   CONST
     Pi   : Extended = 3.14159265358979;
   VAR
     SME1,SME2,SME3, V1,V2a,V2b,V3, a1,a2,a3  : Extended;
   BEGIN
     { --------------------  Initialize values   ------------------ }
     a1 := (r1*(1.0+e1*Cos(Nu1))) / (1.0 - e1*e1 );
     a2 := ( R1 + R3 ) / 2.0;
     a3 := (r3*(1.0+e3*Cos(Nu3))) / (1.0 - e3*e3 );
     SME1:= -1.0 / (2.0*a1);
     SME2:= -1.0 / (2.0*a2);
     SME3:= -1.0 / (2.0*a3);
     DeltaVa:= 0.0;
     DeltaVb:= 0.0;
     TOF:= 0.0;

     IF ( e1 < 1.0 ) or ( e3 < 1.0 ) THEN
       BEGIN

     { ----------------  Find Delta v at point a  ---------------- }
         V1 := SQRT( 2.0*( (1.0/R1) + SME1 ) );
         V2a:= SQRT( 2.0*( (1.0/R1) + SME2 ) );
         DeltaVa:= ABS( V2a - V1 );

     { ----------------  Find Delta v at point b  ---------------- }
         V3 := SQRT( 2.0*( (1.0/R3) + SME3 ) );
         V2b:= SQRT( 2.0*( (1.0/R3) + SME2 ) );
         DeltaVb:= ABS( V3 - V2b );

     { ---------------  Find Transfer Time of Flight  ------------ }
         TOF:= Pi * SQRT( A2*A2*A2 );

         IF Show = 'Y' THEN
           BEGIN
             WriteLn( '  a2  ',a2:10:6,' DU' );
             WriteLn( '  V1  ',v1:10:6 );
             WriteLn( '  V2a ',v2a:10:6,' V2b ',v2b:10:6 );
             WriteLn( '  V3  ',v3:10:6 );
             WriteLn( 'TOTAL ',(DeltaVa+DeltaVb):10:6,' DU/TU' );
           END;
       END;
   END;  { Procedure Hohmann }

{
```

```
}
{ ------------------------------------------------------------------------------
|
|                         PROCEDURE ONETANGENT
|
| This procedure calculates the delta V's for a One Tangent transfer for either
|   circle to circle, or ellipse to ellipse.  The notation used is from the
|   initial orbit (1) at point a, transfer is made to the transfer orbit (2),
|   and to the final orbit (3) at point b.
|
| Algorithm     : Find the parameters for the transfer orbit
|                 Based on the eccentricity, find the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  19 Jun 1989
|
| Inputs        :
|   R1          - Initial position magnitude         DU
|   R3          - Final position magnitude           DU
|   e1          - Eccentricity of first orbit
|   e3          - Eccentricity of final orbit
|   Nu1         - True Anomaly of first orbit        rad
|   Nu2         - True Anomaly of second orbit       rad
|   Nu3         - True Anomaly of final orbit        rad
|
| OutPuts       :
|   DeltaVa     - Change in velocity at point a      DU / TU
|   DeltaVb     - Change in velocity at point b      DU / TU
|   TOF         - Time of Flight for the transfer    TU
|
| Locals        :
|   SME1        - Specfic Mechanical Energy of first orbit       DU2 / TU
|   SME2        - Specfic Mechanical Energy of transfer orbit    DU2 / TU
|   SME3        - Specfic Mechanical Energy of final orbit       DU2 / TU
|   V1          - Velocity of 1st orbit at point a               DU / TU
|   V2a         - Velocity of transfer orbit at point a          DU / TU
|   V2b         - Velocity of transfer orbit at point b          DU / TU
|   V3          - Velocity of final orbit at point b             DU / TU
|   e2          - Eccentricity of second orbit
|   a1          - Semi Major Axis of first orbit                 DU
|   a2          - Semi Major Axis of Transfer orbit              DU
|   a3          - Semi Major Axis of final orbit                 DU
|   E           - Eccentric anomaly of transfer orbit at point b rad
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
| References    :
|   BMW           pg. 163-166
|
 ------------------------------------------------------------------------------ }
{
```

```
}
PROCEDURE OneTangent           ( R1,R3,e1,e3,Nu1,Nu2,Nu3  : Extended;
                                 VAR Deltava,Deltavb,TOF  : Extended );
   VAR
      SME1,SME2,SME3, V1,V2a,V2b,V3, e2,a1,a2,a3, Phi2b,Phi3, E, Sinv,Cosv       : Extend
   BEGIN
     { -------------------- Initialize values   ------------------ }
     a1 := (r1*(1.0+e1*Cos(Nu1))) / (1.0 - e1*e1 );
     e2 := ( r3-r1 ) / ( -r3*Cos(Nu2)+Cos(Nu1)*r1 );  { Cos(Nu1) determines the sign}
     IF ABS( e2-1.0 ) > 0.000001 THEN
       BEGIN
         a2  := (r1*(1.0+e2*Cos(Nu1))) / (1.0 - e2*e2 );
         SME2:= -1.0 / (2.0*a2);
       END
       ELSE
       BEGIN
         a2  := 999999.9;  { Undefined for Parabolic orbit }
         SME2:= 0.0;
       END;
     a3 := (r3*(1.0+e3*Cos(Nu3))) / (1.0 - e3*e3 );
     SME1:= -1.0 / (2.0*a1);
     SME3:= -1.0 / (2.0*a3);

     { ----------------- Find Delta v at point a  ---------------- }
     V1 := SQRT( 2.0*( (1.0/R1) + SME1 ) );
     IF ABS( SME2 ) > 0.000001 THEN
         V2a:= SQRT( 2.0*( (1.0/R1) + SME2 ) )
       ELSE
         V2a:= SQRT( 2.0*(1.0/R1) );
     DeltaVa:= ABS( V2a - V1 );

     { ----------------- Find Delta v at point b  ---------------- }
     V3 := SQRT( 2.0*( (1.0/R3) + SME3 ) );
     IF ABS( SME2 ) > 0.000001 THEN
         V2b:= SQRT( 2.0*( (1.0/R3) + SME2 ) )
       ELSE
         V2b:= SQRT( 2.0*(1.0/R3) );

     Phi2b:= ArcTAN( ( e2*Sin(Nu2) ) / ( 1.0 + e2*Cos(Nu2) ) );
     Phi3 := ArcTAN( ( e3*Sin(Nu3) ) / ( 1.0 + e3*Cos(Nu3) ) );
     DeltaVb:= SQRT( V2b*V2b + V3*V3 - 2.0*V2b*V3*Cos( Phi2b-Phi3 ) );

     { ---------------- Find Transfer Time of Flight  ------------- }
     IF e2 < 0.9999 THEN
       BEGIN
         Sinv:= ( SQRT( 1.0-e2*e2 )*Sin(Nu2) ) / ( 1.0 + e2*Cos(Nu2) );
         Cosv:= ( e2+Cos(Nu2) ) / ( 1.0 + e2*Cos(Nu2) );
         E   := ATAN2( Sinv,Cosv );
         TOF := SQRT( A2*A2*A2 ) * ( E - e2*Sin(E) );
       END
       ELSE
       BEGIN
         IF ABS( e2-1.0 ) < 0.000001 THEN
           BEGIN
             { Parabolic TOF }
           END
           ELSE
           BEGIN
             { Hyperbolic TOF }
           END;
       END;

{}       IF Show = 'Y' THEN
{}         BEGIN
{}           WriteLn( '  a2   ',a2:10:6,' DU' );
{}           WriteLn( '  V1   ',v1:10:6 );
{}           WriteLn( '  V2a  ',v2a:10:6,' V2b ',v2b:10:6 );
{}           WriteLn( '  V3   ',v3:10:6 );
{}           WriteLn( 'TOTAL ',(DeltaVa+DeltaVb):10:6,' DU/TU' );
{}           WriteLn( ' e2    ',e2:10:6,' E = ',E*57.2955:10:6,' a2 ',a2:10:6 );
{}           WriteLn( ' Phi2 ',Phi2b*57.2955:10:6,' Phi3 = ',Phi3*57.2955:10:6 );
{}         END;

   END;  { Procedure OneTangent }
{
```

```
}
{ -------------------------------------------------------------------------
|
|                      PROCEDURE GENERALCOPLANAR
|
| This procedure calculates the delta v's for a general coplanar transfer for
|   either circle to circle, or ellipse to ellipse.  The notation used is from
|   the initial orbit (1) at point a, transfer is made to the transfer orbit (2),
|   and to the final orbit (3) at point b.
|
| Algorithm     :
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  14 Mar 1989
|
| Inputs        :
|   R1          - Initial position magnitude                    DU
|   R3          - Final position magnitude                      DU
|   e1          - Eccentricity of first orbit
|   e3          - Eccentricity of final orbit
|   Nu1         - True Anomaly of first orbit                   rad
|   Nu3         - True Anomaly of final orbit                   rad
|
| OutPuts       :
|   DeltaVa     - Change in velocity at point a                DU / TU
|   DeltaVb     - Change in velocity at point b                DU / TU
|   TOF         - Time of Flight for the transfer                TU
|
| Locals        :
|   SME1        - Specfic Mechanical Energy of first orbit      DU2 / TU
|   SME2        - Specfic Mechanical Energy of transfer orbit   DU2 / TU
|   SME3        - Specfic Mechanical Energy of final orbit      DU2 / TU
|   V1          - Velocity of 1st orbit at point a              DU / TU
|   V2a         - Velocity of transfer orbit at point a         DU / TU
|   V2b         - Velocity of transfer orbit at point b         DU / TU
|   V3          - Velocity of final orbit at point b            DU / TU
|   a1          - Semi Major Axis of first orbit                DU
|   a2          - Semi Major Axis of Transfer orbit             DU
|   a3          - Semi Major Axis of final orbit                DU
|   E           - Eccentric anomaly of transfer orbit at point b rad
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
| References    :
|   BMW           pg.
|
 ------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE GeneralCoplanar       ( R1,R3,e1,e2,e3,Nu1,Nu2a,Nu2b,Nu3       : Extended;
                                  VAR Deltava,Deltavb,TOF                : Extended );
   VAR
      SME1,SME2,SME3, V1,V2a,V2b,V3, a1,a2,a3,Phi1,Phi2a,Phi2b,Phi3,
      E,Eo,Sinv,Cosv    : Extended;
   BEGIN
     { -------------------- Initialize values  ------------------ }
     a1 := (r1*(1.0+e1*Cos(Nu1))) / (1.0 - e1*e1 );
     IF ABS( e2-1.0 ) > 0.000001 THEN
       BEGIN
         a2  := (r1*(1.0+e2*Cos(Nu2a))) / (1.0 - e2*e2 );
         SME2:= -1.0 / (2.0*a2);
       END
       ELSE
       BEGIN
         a2  := 999999.9;  { Undefined for Parabolic orbit }
         SME2:= 0.0;
       END;
     a3 := (r3*(1.0+e3*Cos(Nu3))) / (1.0 - e3*e3 );
     SME1:= -1.0 / (2.0*a1);
     SME3:= -1.0 / (2.0*a3);

     { ----------------- Find Delta v at point a ---------------- }
     V1 := SQRT( 2.0*( (1.0/R1) + SME1 ) );
     V2a:= SQRT( 2.0*( (1.0/R1) + SME2 ) );
     Phi2a:= ArcTAN( ( e2*Sin(Nu2a) ) / ( 1.0 + e2*Cos(Nu2a) ) );
     Phi1 := ArcTAN( ( e1*Sin(Nu1) ) / ( 1.0 + e1*Cos(Nu1) ) );
     DeltaVa:= SQRT( V2a*V2a + V1*V1 - 2.0*V2a*V1*Cos( Phi2a-Phi1 ) );

     { ----------------- Find Delta v at point b ---------------- }
     V3 := SQRT( 2.0*( (1.0/R3) + SME3 ) );
     V2b:= SQRT( 2.0*( (1.0/R3) + SME2 ) );
     Phi2b:= ArcTAN( ( e2*Sin(Nu2b) ) / ( 1.0 + e2*Cos(Nu2b) ) );
     Phi3 := ArcTAN( ( e3*Sin(Nu3) ) / ( 1.0 + e3*Cos(Nu3) ) );
     DeltaVb:= SQRT( V2b*V2b + V3*V3 - 2.0*V2b*V3*Cos( Phi2b-Phi3 ) );

     { ---------------- Find Transfer Time of Flight ------------ }
     Sinv:= ( SQRT( 1.0-e2*e2 )*Sin(Nu2b) ) / ( 1.0 + e2*Cos(Nu2b) );
     Cosv:= ( e2+Cos(Nu2b) ) / ( 1.0 + e2*Cos(Nu2b) );
     E:= ATAN2( Sinv,Cosv );
     Sinv:= ( SQRT( 1.0-e2*e2 )*Sin(Nu2a) ) / ( 1.0 + e2*Cos(Nu2a) );
     Cosv:= ( e2+Cos(Nu2a) ) / ( 1.0 + e2*Cos(Nu2a) );
     Eo:= ATAN2( Sinv,Cosv );
     TOF:= SQRT( A2*A2*A2 ) * ( (E - e2*Sin(E)) - (Eo - e2*Sin(Eo)) );

   END;  { Procedure GeneralCoplanar }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE RENDEZVOUS
|
| This procedure calculates parameters for a Hohmann transfer rendezvous.
|
| Algorithm      : Calculate the answer
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109  23 Sep 1988
|
| Inputs         :
|   Rcs1         - Radius of circular orbit interceptor     DU
|   Rcs2         - Radius of circular orbit target          DU
|   PhaseI       - Initial phase angle                      rad
|   NumRevs      - Number of revs to wait
|
| OutPuts        :
|   PhaseF       - Final Phase Angle                        rad
|   WaitTime     - Wait before next intercept opportunity   TU
|
| Locals         :
|   TOFTrans     - Time of flight of transfer orbit         TU
|   ATrans       - Semi-major axis of transfer orbit        DU
|   VelTgt       - Velocity of target                       rad / TU
|   VelInt       - Velocity of interceptor                  rad / TU
|   LeadAng      - Lead Angle                               rad
|
| Constants      :
|   Pi
|
| Coupling       :
|   None.
|
| References     :
|   BMW            pg.
|
 ----------------------------------------------------------------------- }

PROCEDURE Rendezvous           ( Rcs1,Rcs2,PhaseI         : Extended;
                                 NumRevs                  : Integer;
                                 VAR PhaseF,WaitTime      : Extended );
   CONST
     Pi : Extended = 3.14159265358979;
   VAR
     TOFTrans,LeadAng,aTrans,VelTgt,VelInt    : Extended;
   BEGIN
     ATrans   := (Rcs1 + Rcs2) / 2.0;
     TOFTrans:= Pi*SQRT( ATrans*ATrans*ATrans );
     VelInt   := 1.0 / ( SQRT(Rcs1*Rcs1*Rcs1) );
     VelTgt   := 1.0 / ( SQRT(Rcs2*Rcs2*Rcs2) );

     LeadAng := VelTgt * TOFTRans;
     PhaseF   := Pi - LeadAng;
     WaitTime:= ( PhaseI - PhaseF + 2.0*Pi*NumRevs ) / ( VelInt - VelTgt );

{}      IF Show = 'Y' THEN
{}        BEGIN
{}          Writeln( '    A transfer  = ',ATRans:12:8,  ' DU ' );
{}          Writeln( '    TOF Transfer= ',TOFTRans:12:8,' TU ' );
{}          WriteLn( '    VelTgt      = ',VelTgt:12:8,  ' rad/TU' );
{}          WriteLn( '    VelInt      = ',VelInt:12:8,  ' rad/TU' );
{}          Writeln( '    Lead Angle  = ',LeadAng*57.29578:12:8,' ' );
{}        END;

   END;  { Procedure Rendezvous }
{
```

```
}
{ -----------------------------------------------------------------------------
|
|                          PROCEDURE INTERPLANETARY
|
|  This procedure calculates the delta v's for an interplanetary mission.  The
|    transfer assumes circular orbits for each of the planets.  Notice the
|    units are all metric since this procedure is designed for ANY planet and
|    sun system.  This eliminates having knowledge of cannonical units for
|    each planet in the calculations.
|
|  Algorithm     : Calculate the answer
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
|
|  Inputs        :
|    R1          - Radius of planet 1 from sun                     km
|    R2          - Radius of planet 2 from sun                     km
|    Rbo         - Radius at burnout about planet 1                km
|    Rimpact     - Radius at impact on planet 2                    km
|    Mu1         - Gravitational parameter of planet 1             km3/s2
|    Mut         - Gravitational parameter of planet Sun           km3/s2
|    Mu2         - Gravitational parameter of planet 2             km3/s2
|
|  OutPuts       :
|    DeltaV1     - Hyperbolic Excess velocity at planet 1 SOI      km/s
|    DeltaV2     - Hyperbolic Excess velocity at planet 2 SOI      km/s
|    Vbo         - Burnout velocity at planet 1                    km/s
|    Vretro      - Retro velocity at surface of planet 2           km/s
|
|  Locals        :
|    SME1        - Specfic Mechanical Energy of 1st orbit          Km2/s
|    SMEt        - Specfic Mechanical Energy of transfer orbit     Km2/s
|    SME2        - Specfic Mechanical Energy of 2nd orbit          Km2/s
|    Vcs1        - Velocity of 1st orbit at delta v 1 point        Km/s
|    Vcs2        - Velocity of 2nd orbit at delta v 2 point        Km/s
|    Vt1         - Velocity of Transfer orbit at delta v 1 point   Km/s
|    Vt2         - Velocity of Transfer orbit at delta v 2 point   Km/s
|    A           - Semi Major Axis of Transfer orbit               Km
|
|  Constants     :
|    None.
|
|  Coupling      :
|    None.
|
|  References    :
|    BMW           pg.
|
 ---------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE Interplanetary        ( R1,R2,Rbo,Rimpact,Mu1,Mut,Mu2       : Extended;
                                  VAR Deltav1,Deltav2,Vbo,Vretro      : Extended );
   VAR
      SME1,SME2,SMEt, Vcs1, Vcs2, Vt1, Vt2, A,TP        : Extended;
   BEGIN
     { - Find a, SME, apogee and perigee velocities of transfer orbit - }
     A    := (R1+R2) / 2.0;
     SMEt:= -Mut/ (2.0*A);
     Vt1 := SQRT( 2.0*( (Mut/R1) + SMEt ) );
     Vt2 := SQRT( 2.0*( (Mut/R2) + SMEt ) );

     { ---- Find circular velocities of launch and target planet ---- }
     Vcs1:= SQRT( Mut/R1 );
     Vcs2:= SQRT( Mut/R2 );

     { ---- Find delta velocities for Hohmann transfer portion  ----- }
     DeltaV1:= ABS( Vt1 - Vcs1 );
     DeltaV2:= ABS( Vcs2 - Vt2 );

     { - Find SME and burnout/impact vel of launch / target planets - }
     SME1  := Deltav1*DeltaV1 / 2.0;
     SME2  := Deltav2*DeltaV2 / 2.0;
     Vbo   := SQRT( 2.0*( (Mu1/Rbo) + SME1 ) );
     Vretro:= SQRT( 2.0*( (Mu2/Rimpact) + SME2 ) );

{}   IF Show = 'Y' THEN
{}     BEGIN
{}       TP:= Pi*SQRT( a*a*a/Mut );  { Transfer Period in secs }
{}       WriteLn( '     Transfer Period = ',TP/3.1536E07:8:3,' yrs  or  ',TP/86400.0:8:3,'
{}       Writeln;
{}       WriteLn( 'Vcs  km/s':19,vcs1:9:4,' ':10,vcs2:9:4);
{}       WriteLn( ' Vt  km/s':19,vt1:9:4,' ':10,vt2:9:4 );
{}       WriteLn( 'SME km2/s2':19,SME1:9:4,' ',SMEt:9:3,SME2:9:4 );
{}     END;

   END;  { Procedure Interplanetary }
{
```

```
}
{ --------------------------------------------------------------------------
|
|                          PROCEDURE REENTRY
|
| This procedure calculates various reentry parameters using the
|   Allen & Eggars approximations.
|
| Algorithm     : Find the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  14 Apr 1989
|
| Inputs        :
|   Vre         - Reentry Velocity                        m/s
|   PhiRe       - Reentry Flight Path Angle               rad
|   BC          - Ballistic Coefficient                   kg/m2
|   h           - Altitude                                km
|
| Outputs       :
|   V           - Velocity                                km/s
|   Decl        - Decleration                             g's
|   MaxDecl     - Maximum Decleration                     g's
|
| Locals        :
|   grav        - Temporary variable to hold Weight component
|   Rho         - Atmospheric density                     kg/m3
|
| Constants     :
|   ScaleHt     - Scale height used to exponentially model atmosp 1.0/7.315
|
| Coupling      :
|   None.
|
| References    :
|   None.
|
 -------------------------------------------------------------------------- }

PROCEDURE Reentry             ( Vre,PhiRe,BC,h          : Extended;
                                VAR V,Decl,MaxDecl      : Extended );

   VAR
      ScaleHt,grav,Rho   : Extended;
   BEGIN
     ScaleHt:= 1.0/7.315;

     Rho:= 1.225*EXP( -ScaleHt*h );
     V  := Vre * EXP( (1000.0*Rho ) / (2.0*BC*ScaleHt*Sin(PhiRe)) );
     grav:= 9.80*Sin(PhiRe);
     Decl:= ((-0.5*Rho*V*V) / BC ) + grav;
     Decl:= Decl/9.80;

     MaxDecl:= (-0.5*ScaleHt*Vre*Vre*Sin(Phire)) / (9.80*EXP(1.0));
     MaxDecl:= MaxDecl/9.80;
   END;  { Procedure Reentry }

{
```

```
}
{ ------------------------------------------------------------------------------
|
|                          PROCEDURE HILLSR
|
| This procedure calculates various position information for Hills equations.
|   Notice the XYZ system used has Y Colinear with Target Position vector,
|   Z normal to target orbit plane, and x in direction of velocity.
|
| Algorithm      : Find the answer
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   8 May 1989
|
| Inputs         :
|   R            - Initial Position vector of INT          DU
|   V            - Initial Velocity Vector of INT          DU / TU
|   Alt          - Altitude of TGT satellite               DU
|   T            - Desired Time                            TU
|
| Outputs        :
|   R1           - Final Position vector of INT            DU
|   V1           - Final Velocity Vector of INT            DU / TU
|
| Locals         :
|
| Constants      :
|
| Coupling       :
|   None.
|
| References     :
|   Kaplan         pg 108 - 115
|
 ------------------------------------------------------------------------------ }

PROCEDURE HillsR                  ( r,v                          : Vector;
                                    Alt,t                        : Extended;
                                    VAR R1,V1                    : Vector    );

   VAR
      SinNt,CosNt,Omega,nt,Radius        : Extended;
   BEGIN
     { ---------------- Initialize the orbit elemnts --------------- }
     Radius:= 1.0 + Alt;
     Omega:= SQRT( 1.0 / Radius );
     nt    := Omega*t;
     CosNt:= Cos( nt );
     SinNt:= Sin( nt );

     { ---------------- Determine new positions  ------------------ }
     R1[1]:= ( 2.0*V[2]/Omega ) * CosNt +
             ( (4.0*V[1]/Omega) + 6.0*R[2] ) * SinNt +
             ( R[1] - (2.0*V[2]/Omega) ) -
             ( 3.0*V[1] + 6.0*Omega*R[2] )*t;
     R1[2]:= ( V[2]/Omega ) * SinNt -
             ( (2.0*V[1]/Omega) + 3.0*R[2] ) * CosNt +
             ( (2.0*V[1]/Omega) + 4.0*R[2] );
     R1[3]:= R[3]*CosNt + (V[3]/Omega)*SinNt;

     { ---------------- Determine new velocities  ------------------ }
     V1[2]:= 0.0;
     V1[1]:= 0.0;
     V1[3]:= 0.0;

   END;  { Procedure HillsR }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                               PROCEDURE HILLSV
|
|  This procedure calculates initial velocity for Hills equations.
|    Notice the XYZ system used has Y Colinear with Target Position vector,
|    Z normal to target orbit plane, and x in direction of velocity.
|
|  Algorithm     : Check for a divide by zero, then
|                  Find the answer
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   8 May 1989
|
|  Inputs        :
|    R           - Initial Position vector of INT          DU
|    Alt         - Altitude of TGT satellite               DU
|    T           - Desired Time                            TU
|
|  Outputs       :
|    V           - Initial Velocity Vector of INT          DU / TU
|
|  Locals        :
|
|  Constants     :
|
|  Coupling      :
|    None.
|
|  References    :
|    Kaplan        pg 108 - 115
|
 ------------------------------------------------------------------------ }

PROCEDURE HillsV                 ( r                        : Vector;
                                   Alt,t                    : Extended;
                                   VAR V                    : Vector    );

   VAR
     Numer,Denom,SinNt,CosNt,Omega,nt,Radius          : Extended;
   BEGIN
     { ---------------- Initialize the orbit elemnts --------------- }
     Radius:= 1.0 + Alt;
     Omega:= SQRT( 1.0 / Radius );
     nt   := Omega*t;
     CosNt:= Cos( nt );
     SinNt:= Sin( nt );

     { ---------------- Determine initial Velocity ----------------- }
     Numer:= ( (6.0*r[2]*(nt-SINnT)-r[1])*Omega*Sinnt-2.0*Omega*r[2]*
              (4.0-3.0*Cosnt)*(1.0-COSnt) );
     Denom:= (4.0*Sinnt-3.0*nt)*Sinnt + 4.0*( 1.0-CosNt ) * ( 1.0-CosNt );

     IF ABS( Denom ) > 0.000001 THEN
         V[1]:= Numer / Denom
       ELSE
         V[1]:= 0.0;
     IF ABS( SinNt ) > 0.000001 THEN
         V[2]:= -( Omega*r[2]*(4.0-3.0*Cosnt)+2.0*(1.0-Cosnt)*v[1] ) /
                 ( SinNt )
       ELSE
         V[2]:= 0.0;
     V[3]:= 0.0;

   END;  { Procedure HillsV }
{
```

```
}
{ -----------------------------------------------------------------------------
|
|                              PROCEDURE TARGET
|
|  This procedure accomplishes the targeting problem using KEPLER and GAUSS.
|
|  Algorithm     : Propogate the target forward
|                  Find the intercept trajectory
|                  Calculate the change in velocity required
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  8 Jun 1990
|
|  Inputs        :
|    RInt        - Initial Position vector of Interceptor     DU
|    VInt        - Initial Velocity vector of Interceptor     DU/TU
|    RTgt        - Initial Position vector of Target          DU
|    VTgt        - Initial Velocity vector of Target          DU/TU
|    dm          - Direction of Motion for Gauss              'L','S'
|    TOF         - Time of flight to the intercept            TU
|
|  Outputs       :
|    V1t         - Initial Transfer Velocity vector           DU/TU
|    V2t         - Final Transfer Velocity vector             DU/TU
|    DV1         - Initial Change Velocity vector             DU/TU
|    DV2         - Final Change Velocity vector               DU/TU
|
|    Direc       - Direction of transfer,DNu ls or gt Pi      'L','S'
|
|  Locals        :
|    TransNormal - Cross product result of transfer orbit     DU
|    IntNormal   - Cross product result of interceptor orbit  DU
|    R1Tgt       - Position vector after TOF of Target        DU
|    V1Tgt       - Velocity vector after TOF of Target        DU/TU
|    RIRT        - RInt[4] * R1Tgt[4]
|    CosDeltaNu  - Cosine of DeltaNu                          rad
|    SinDeltaNu  - Sine of DeltaNu                            rad
|    DeltaNu     - DeltaNu, angle between position vectors    rad
|
|  Constants     :
|    None
|
|  Coupling      :
|    CROSS         Cross product of two vectors
|    KEPLER        Find R and V at future time
|    GAUSS         Find velocity vectors at each end of transfer
|    LNCOM2        Linear combination of two vectors and constants
|    DOT           DOT product of two vectors
|
|  References    :
|    None.
|
 ------------------------------------------------------------------------------ }

PROCEDURE TARGET               ( RInt,VInt,RTgt,VTgt       : Vector;
                                 Dm                        : CHAR;
                                 TOF                       : Extended;
                                 VAR V1t,V2t,DV1,DV2       : Vector  );
   VAR
     IntNormal, TransNormal, R1Tgt, V1Tgt           : Vector;
     Temp, RIRT, CosDeltaNu, SinDeltaNu, DeltaNu     : EXTENDED;
   BEGIN
     { ----------- Propogate target forward by TOF ----------------- }
     KEPLER( RTgt,VTgt,TOF,  R1Tgt,V1Tgt );

     { ----------- Calculate transfer orbit between r's ------------ }
     GAUSS( RInt,R1Tgt,dm,TOF,  V1t,V2t );

     LNCOM2( -1.0, 1.0,VInt, V1t,  DV1 );
     LNCOM2(  1.0,-1.0,V1Tgt,V2t,  DV2 );

     IF V1t[4] < 0.00001 THEN
       DV1[4]:= 100.0;
   END;  { Procedure Target }
{
```

```
}
{ -----------------------------------------------------------------------------
|
|                            PROCEDURE PKEPLEP
|
|  This procedure propagates a satellite's position and velocity vector over
|    a given time period accounting for perturbations caused by J2.  The
|    satellite's original position and velocity vectors are inputed together
|    with the time the elements are to be propagated for.  The updated position
|    and velocity vectors are then output.
|
|  Algorithm     : Find the value of the perturbations
|                  Determine the type of orbit
|                  Update the appropriate parameters
|                  Find the new position and velocity vectors
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   6 Jan 1990
|
|  Inputs        :
|    R           - original position vector                DU
|    V           - original velocity vector                DU/TU
|    DeltaT      - time for which orbital elements are to TU
|
|  Outputs       :
|    R1          - updated position vector                 DU
|    V1          - updated velocity vector                 DU/TU
|
|  Locals        :
|    P           - Semi-paramter                           DU
|    A           - semimajor axis                          DU
|    E           - eccentricity
|    Inc         - inclination                             rad
|    Argp        - argument of periapsis                   rad
|    ArgpDot     - change in argument of periapsis         rad/TU
|    Omega       - longitude of the ascending node         rad
|    OmegaDot    - change in Omega                         rad
|    E0          - eccentric anomaly                       rad
|    E1          - eccentric anomaly                       rad
|    M           - mean anomaly                            rad/TU
|    MDot        - change in mean anomaly                  rad/TU
|    Uo          - argument of latitude                    rad
|    UDot        - change in argument of latitude          rad/TU
|    Lo          - true longitude of vehicle               rad
|    LDot        - change in the true longitude            rad/TU
|    CapPio      - longitude of periapsis                  rad
|    CapPioDot   - longitude of periapsis change           rad/TU
|    N           - mean angular motion                     rad/TU
|    NUo         - true anomaly                            rad
|    J2oP2       - J2 over p sqyared
|    Sinv,Cosv   - Sine and Cosine of Nu
|
|  Constants     :
|    Pi          -
|    TwoPi       -
|    J2          - J2 constant from the Earth's geopotential function
|    Small       - Tolerance
|
|  Coupling:
|    ELORB       - Orbit Elements from position and Velocity vectors
|    RANDV       - Position and Velocity Vectors from orbit elements
|    NewtonR     - Newton Rhapson to find Nu and Eccentric anomaly
|    RealMod     - Real MOD operation
|
|  References    :
|    Escobal       pg 369. Dot terms
|    BMW           pg
|
 ----------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE PKepler             ( Ro,Vo                      : Vector;
                                DeltaT                     : Extended;
                                VAR R,V                    : Vector    );
   CONST
     TwoPi : Extended =  6.28318530717959;
     Pi    : Extended =  3.14159265358979;
     J2    : Extended =  0.00108263;
     Small : Extended =  0.000001;
   VAR
     P,A,E,Inc,Omega,Argp,Nuo,M,Uo,Lo,CapPio,OmegaDot,E0,
     ArgpDot,MDot,UDot,LDot,CapPiDot,N,J2oP2,NBar       : Extended;
     TypeOrbit       : STRING[2];
  BEGIN
    ELORB( Ro,Vo,P,A,E,Inc,Omega,Argp,Nuo,M,Uo,Lo,CapPio);
    n:= SQRT(1.0/(A*A*A));

    { ------------- Find the value of J2 perturbations ----------- }
    J2oP2:= (1.5*J2) / (p*p);
    NBar:= n*( 1.0 + J2oP2*SQRT(1.0-e*e)* (1.0 - 1.5*Sin(Inc)*Sin(Inc)) );
    OmegaDot:= -J2oP2 * Cos(Inc) * NBar;
    ArgpDot:=  J2oP2 * (2.0-2.5*Sin(Inc)*Sin(Inc)) * Nbar;
    MDot   :=  NBar;
{      EDot     = -(4.0/3.0) * (1.0-E) * (NDot/Nbar)  Drag Terms}

    { --------- Determine type of orbit for later use ---------- }
          { --- Elliptical, Parabolic, Hyperbolic Inclined --- }
       TypeOrbit:= 'EI';

       IF E < Small THEN
         { ------------- Circular Equatorial  -------------- }
           IF ( Inc < Small ) or ( ABS(Inc-Pi) < Small ) THEN
             TypeOrbit:= 'CE'
           ELSE
         { ------------- Circular Inclined    -------------- }
             TypeOrbit:= 'CI'
         ELSE
         { -- Elliptical, Parabolic, Hyperbolic Equatorial -- }
           IF ( Inc < Small ) or ( ABS(Inc-Pi) < Small ) THEN
             TypeOrbit:= 'EE';

    { ----- Update the orbital elements for each orbit type ------ }
    { ----------------- Elliptical - Inclined ------------------- }
      IF TypeOrbit = 'EI' THEN
        BEGIN
          Omega:= Omega + OmegaDot * DeltaT;
          Omega:= REALMOD(Omega, TwoPi);
          Argp := Argp  + ArgpDot  * DeltaT;
          Argp := REALMOD(Argp, TwoPi);
          M    := M     + MDOT     * DeltaT;
          M    := REALMOD(M, TwoPi);
          NewtonR( e,m,  e0,Nuo );
        END;
    { ------------------ Circular Inclined ---------------------- }
      IF TypeOrbit = 'CI' THEN
        BEGIN
          Omega:= Omega + OmegaDot * DeltaT;
          Omega:= REALMOD(Omega, TwoPi);
          UDot := ArgpDot + MDot;
          Uo   := Uo    + UDot     * DeltaT;
          Uo   := REALMOD(Uo, TwoPi);
        END;
    { ---------------- Elliptical - Equatorial ------------------ }
      IF TypeOrbit = 'EE' THEN
        BEGIN
          CapPiDot:= OmegaDot + ArgpDot;
          CapPio := CapPio    + CapPiDot * DeltaT;
          CapPio := REALMOD(CapPio, TwoPi);
          M      := M       + MDOT     * DeltaT;
          M      := REALMOD(M, TwoPi);
          NewtonR( e,m,  e0,Nuo );
        END;
    { ----------------- Circular - Equatorial ------------------- }
      IF TypeOrbit = 'CE' THEN
        BEGIN
          LDot:= OmegaDot + ArgpDot + MDot;
          Lo := Lo       + LDot      * DeltaT;
          Lo := REALMOD(Lo, TwoPi);
        END;

    { ------------- Use RANDV to find new vectors --------------- }
      RANDV( P,E,Inc,Omega,Argp,Nuo,Uo,Lo,CapPio,  R,V );

  END; { Procedure PKepler }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                         PROCEDURE J2DRAGPERT
|
| This procedure calculates the perturbations for the predict problem
|   involving secular rates of change resulting from J2 and Drag only.
|
| Algorithm     : Find the startup values
|                 Calculate the dot terms
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  14 Mar 1989
|
| Inputs        :
|   Inc         - Inclination                          rad
|   e           - Eccentricity
|   N           - Mean Motion                          rad/TU
|   NDot        - Mean Motion rate                     rad / 2TU2
|
| Outputs       :
|   OmegaDot    - Long of Asc Node rate                rad / TU
|   ArgpDot     - Argument of perigee rate             rad / TU
|   EDot        - Eccentricity rate                    / TU
|
| Locals        :
|   P           - Semi-parameter                       DU
|   A           - Semi-major axis                      DU
|
| Constants     :
|   J2            J2 zonal harmonic
|
| Coupling      :
|   POWER         Raise a base to a power
|
| References    :
|   Escobal       pg. 369
|   O'Keefe et al., Astronomical J, Vol 64 num 7, pg. 247 for Edot
|
 ------------------------------------------------------------------------ }

PROCEDURE J2DragPert    ( Inc,E,N,NDot                   : Extended;
                          VAR OmegaDOT,ArgpDOT,EDOT       : Extended );
   CONST
      J2  : Extended =  0.00108263;
   VAR
      P,A,NBar : Extended;
   BEGIN
     a    := Power( 1.0/n , 2.0/3.0 );
     p    := a*(1.0-e*e);
     NBar:= n*( 1.0+1.5*J2*(SQRT(1.0-e*e)/(p*p))*(1.0-1.5*Sin(inc)*Sin(inc) ));

     { ------------------------ Find dot terms  -------------------- }
     OmegaDot:= -1.5*( J2/(p*p) ) * Cos(inc) * NBar;
     ArgpDot := 1.5*( J2/(p*p) ) * (2.0-2.5*Sin(inc)*Sin(inc)) * Nbar;
     EDot    := -(4.0/3.0) * (1.0-E) * (NDot/Nbar);

   END; { Procedure J2DragPert  }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                              PROCEDURE PREDICT
|
|  This procedure determines the azimuth and elevation for the viewing
|    of a staellite from a known ground site.  Notice the Julian Date is left
|    in it's usual DAYS format since the dot terms are input as radians per
|    day, thus no extra need for conversion.  The Julian Date also facilitates
|    finding the site position vector.  Also observe RANDV is not used since
|    this would merely accomplish extra calculations.  The iteration is left
|    out to allow the user to set up his own loop to look for sighting times.
|
|  Algorithm     :
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   11 Dec 1990
|
|  Inputs        :
|    JD          - Julian Date of desired observation      Day
|    JDEpoch     - Julian date of epoch for satellite      Day
|    No          - Epoch Mean motion                       rad/day
|    Ndot        - Epcoh Half Mean Motion Rate             rad/2day2
|    Eo          - Epoch Eccentricity
|    Edot        - Epoch Eccentricity rate                 /day
|    Inco        - Epoch Inclination                       rad
|    Omegao      - Epoch Lon of Asc node                   rad
|    OmegaDot    - Epoch Lon of Asc Node rate              rad/day
|    Argpo       - Epoch Argument of perigee               rad
|    ArgpDot     - Epoch Argument of perigee rate          rad/day
|    Mo          - Epoch Mean Anomaly                      rad
|    Lat         - Geodetic Latitude of site               rad
|    Lon         - Longitude of site                       rad
|    Alt         - Altitude of site                        DU
|  OutPuts       :
|    Rho         - Range from site to satellite            DU
|    Az          - Azimuth                                 rad
|    El          - Elevation                               rad
|    RtAsc       - Right ascension                         rad
|    Decl        - Declination                             rad
|    Vis         - Visibility  'Radar Sun', 'Eye', 'Radar Nite', 'Not Visible'
|  Locals        :
|    Variable o  - denotes the epoch value, while no o is current
|    Dt          - Change in time from Epoch to desired t day
|    A           - Semi major axis                         DU
|    E0          - Eccentric Anomaly                       rad
|    Nu          - True Anomaly                            rad
|    LST         - Local Sidereal Time                     rad
|    GST         - Greenwich Sidereal Time                 rad
|    Temp        - Temporary Real value
|    SRtAsc      - Suns Right ascension                    rad
|    SDecl       - Suns Declination                        rad
|    Theta       - Angle between IJK Sun and Satellite vecrad
|    Dist        - Ppdculr distance of satellite from RSunDU
|    Small       - Tolerance of small values
|    R           - IJK Satellite vector                    DU
|    RS          - IJK Site Vector                         DU
|    VS          - Site Velocity vector                    DU/TU
|    RhoVec      - Site to satellite vector in SEZ         DU
|    TempVec     - Temporary vector
|    RHoV        - Site to satellite vector in IJK         DU
|    RSun        - Sun vector                              AU
|    C           - Temporary Vector
|
|  Constants     :
|    Pi          -                                         3.14159265358979
|    HalfPi      -                                         1.57079632679490
|    TwoPi       -                                         6.28318530717959
|    Rad         - Degrees per radian                       57.29577951308230
|    TUDay       - Days in one TU                          0.00933809102919444
|    AUDU        - DUs in 1 AU                             23455.07
|
|  Coupling      :
|    SUN           Position vector of Sun
|    MAG           Magnitude of a vector
|    DOT           Dot product of two vectors
|    CROSS         Cross Product of two vectors
|    ROT1,ROT2,ROT3 Rotations about 1st, 2nd and 3rd axis
|    SITE          Site Vector
|    LSTime        Local Sidereal Time
|    NewtonR       Iterate to find Eccentric Anomaly
|    ATAN2         Arc Tangent function which resolves quadrants
|  References    :
|    Escobal       pg. 369
 --------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE Predict            ( JD,JDEpoch,no,Ndot,Eo,Edot,inco,
                               Omegao,OmegaDot,Argpo,ArgpDot,Mo: Extended;
                               Lat,Lon,Alt                    : Extended;
                               VAR Rho,Az,El,Rtasc,Decl       : Extended;
                               VAR Vis                        : Strll    );
   CONST
      Small  : Extended =  0.000001;
      Pi     : Extended =  3.14159265358979;
      HalfPi : Extended =  1.57079632679490;
      TwoPi  : Extended =  6.28318530717959;
      Rad    : Extended = 57.2957795130823;
      TUDay  : Extended =  0.00933809102919444;
      AUDU   : Extended = 23455.07;
   VAR
      Dt,a,E0,Nu,LST,GST,Temp,SRtAsc,SDecl,Theta,Dist,
      N,M,E,Omega,Argp                            : Extended;
      Rpqw,R,RS,VS,RhoVec,TempVec,RhoV,RSun,C     : Vector;
      i                                           : Integer;
   BEGIN
     { --------------------- Initialize values   ------------------- }
     Az     :=  0.0;
     El     :=  0.0;
     Rho    :=  0.0;

     { --------------- Update elements to new time  ---------------- }
     Dt    := JD - JDEpoch;
     e     := eo      + EDot*Dt;
     Omega:= Omegao + OmegaDot*Dt;
     Argp  := Argpo  + ArgpDot*Dt;
     M     := Mo     + No*Dt    + NDot*Dt*Dt;
     M     := RealMOD( M,TwoPi );
     N     := No + 2.0*NDot*Dt;        { n is in rad/DAY , ndot is over 2}
     N     := N * TUDay;               { convert n to rad/TU}

     { ------------ Newton Rhapson to find True Anomaly ------------ }
     NewtonR( e,M, E0,Nu );

     { ------------------ Form PQW position vector  ---------------- }
     a:= POWER( 1.0/(N*N) , 1.0/3.0 );

     Rpqw[4]:= ( a*(1.0-e*e) ) / (1.0 + e*Cos( Nu ) );
     Rpqw[1]:= Rpqw[4]*Cos( Nu );
     Rpqw[2]:= Rpqw[4]*Sin( Nu );
     Rpqw[3]:= 0.0;

     { ---------------------- Rotate to IJK ------------------------ }
     ROT3( Rpqw    , -Argp , TempVec );
     ROT1( TempVec, -Inco , TempVec );
     ROT3( TempVec, -Omega,  R       );

     LSTIME( Lon,JD,     Lst,Gst );
     SITE( Lat,Alt,Lst,  RS,VS );

     { ------- Find IJK range vector from site to satellite -------- }
     FOR i:=1 to 3 DO
         RhoV[i]:=  R[i] - RS[i];
     MAG( RhoV );
     Rho:= RhoV[4];

     { ------  Calculate Topocentric Rt Asc and Declination -------- }
     Temp:= SQRT( RhoV[1]*RhoV[1] + RhoV[2]*RhoV[2] );
     IF ABS( RhoV[2] ) < Small THEN
         IF Temp < Small THEN
           BEGIN
             RtAsc:= 999999.1;
           END
           ELSE
             IF RhoV[1] > 0.0 THEN
                 RtAsc:= Pi
               ELSE
                 RtAsc:= 0.0
       ELSE
         RtAsc := ATAN2( RhoV[2]/Temp , RhoV[1]/Temp );
     IF Temp < Small THEN
         Decl:= HalfPi  { Check for case of -90 deg ***** }
       ELSE
         Decl:= ARCSIN( RhoV[3]/RhoV[4] );

     { ---------------------- Rotate to SEZ ------------------------ }
     ROT3( RhoV,      LST    ,  TempVec );
     ROT2( TempVec,HalfPi-Lat,   RhoVec );
{
```

```
}
     { --------------- Check visibility constraints ---------------- }
     { ------------------ Is it above the Horizon ------------------ }
     IF RhoVec[3] > 0.0 THEN
       BEGIN

     { ---------- Is the site in the light, or the dark? ----------- }
         SUN( JD,RSun,SRtAsc,SDecl );
         LNCOM1( AUDU,RSun, RSun );
         IF DOT( RSun, RS ) > 0.0 THEN
             Vis:= 'Radar Sun  '
           ELSE
           BEGIN

     { ---------- Is the satellite in the shadow or not? ----------- }
             CROSS( RSun, R, C );
             Theta:= ArcSin( C[4]/ (RSun[4]*R[4]) );
             Dist:= R[4]*COS( Theta - HalfPi );
             IF Dist > 1.0 THEN
                 Vis:= 'Eye        '
               ELSE
                 Vis:= 'Radar Nite ';
           END;
       END
       ELSE
         Vis:= 'Not Visible';

     { ------------  Calculate Azimuth and Elevation  -------------- }
     Temp:= SQRT( RhoVec[1]*RhoVec[1] + RhoVec[2]*RhoVec[2] );
     IF ABS( RhoVec[2] ) < Small THEN
         IF Temp < Small THEN
           BEGIN
             Write( ^G );
             Az:= 999999.1;
           END
           ELSE
             IF RhoVec[1] > 0.0 THEN
                 Az:= Pi
               ELSE
                 Az:= 0.0
       ELSE
         Az:= ATAN2( RhoVec[2]/Temp , -RhoVec[1]/Temp );
     IF ( Temp < Small ) THEN
         El:= HalfPi
       ELSE
         El:= ArcSin( RhoVec[3]/Rho );

     { ------  Calculate Geocentric Rt Asc and Declination --------- }
{     Temp:= SQRT( R[1]*R[1] + R[2]*R[2] );
|     IF ABS( R[2] ) < Small THEN
|         IF Temp < Small THEN
|           BEGIN
|             Write( ^G );
|             RtAsc:= 999999.1;
|           END
|           ELSE
|             IF R[1] > 0.0 THEN
|                 RtAsc:= Pi
|               ELSE
|                 RtAsc:= 0.0
|       ELSE
|         RtAsc:= ATAN2( R[2]/Temp , R[1]/Temp );
|     IF ( Temp < Small ) THEN
|         Decl:= HalfPi
|       ELSE
|         Decl:= ARCSIN( R[3]/R[4] );
}
   END;  { Procedure Predict }
{
```

```
}
{ ----------------------------------------------------------------------------
|
|                           PROCEDURE DERIV
|
| This procedure calculates the derivative of the two-body state vector for
|   use with the Runge-Kutta algorithm.
|
| Algorithm     : Find the answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  28 Aug 1989
|
| Inputs        :
|   ITime       - Time                                  TU
|   X           - State Vector                          DU, DU/TU
|
| Outputs       :
|   XDot        - Derivative of State Vector            DU/TU,  DU/TU2
|
| Locals        :
|   RCubed      - Cube of R
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
| References    :
|   None.
|
 ---------------------------------------------------------------------------- }

PROCEDURE Deriv                ( ITime                      : Extended;
                                 X                          : Matrix;
                                 VAR XDot                   : Matrix );
   VAR
      R,RCubed : Extended;
   BEGIN
     R:= SQRT( SQR(GetVal(X,1,1)) + SQR(GetVal(X,2,1)) + SQR(GetVal(X,3,1)) );
     RCubed:= R*R*R;

     { -------------------- Velocity Terms ----------------------- }
     AssignVal( XDot,1,1,GetVal(X,4,1) );
     AssignVal( XDot,2,1,GetVal(X,5,1) );
     AssignVal( XDot,3,1,GetVal(X,6,1) );

     { ------------------ Acceleration Terms ---------------------- }
     AssignVal( XDot,4,1,-GetVal(X,1,1) / RCubed );
     AssignVal( XDot,5,1,-GetVal(X,2,1) / RCubed );
     AssignVal( XDot,6,1,-GetVal(X,3,1) / RCubed );

   END;  { Procedure Deriv }
{
```

```
}
{ -----------------------------------------------------------------------
|
|                          PROCEDURE PERTACCEL
|
|  This procedure calculates the actual value of the perturbing acceleration.
|
|  Algorithm     : Setup temporary values
|                  Use a case statement to select which perturbations are added
|                  Note each perturbation adds on to the previous result
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
|
|  Inputs        :
|    R           - Radius vector                              DU
|    V           - Velocity vector                            DU/TU
|    Time        - Initial time                               TU
|    WhichOne    - Which perturbation to calculate            1 2 3 4 5 ...
|    BC          - Ballistic Coefficient                      kg/m2
|
|  Outputs       :
|    APert       - Perturbing acceleration                    DU/TU2
|
|  Locals        :
|    rs2         - Sun radius vector **2
|    rs3         - Sun radius vector **3
|    rm2         - Moon radius vector **2
|    rm3         - Moon radius vector **3
|    r32         - "z" component of Radius vector **2
|    r33         - "z" component of Radius vector **3
|    r34         - "z" component of Radius vector **4
|    r2          - Radius vector **2
|    r3          - Radius vector **3
|    r4          - Radius vector **4
|    r5          - Radius vector **5
|    r7          - Radius vector **7
|    Beta        -
|    Temp        - Temporary Real Value
|    rho         - Atmospheric Density
|    Va          - Relative Velocity Vector                   DU / TU
|    RSun        - Radius Vector to Sun                       AU
|    RMoon       - Radius Vector to Moon                      DU
|    RtAsc       - Right Ascension                            deg
|    Decl        - Declination                                deg
|    i           - Index
|
|  Constants     :
|    J2                                                         0.00108263
|    J3          -                                             -0.00000254
|    J4          -                                             -0.00000161
|    GMS         - Sun Gravitational Parameter DU3/TU2          332952.9364
|    GMM         - Moon Gravitational Parameter DU3/TU2         0.01229997
|    OmegaEarth  - Angular rotation of Earth (Rad/TU)           0.0588335906868878
|    TUDay       - Days in one TU                               0.00933809102919444
|
|  Coupling      :
|    MAG           Magnitude of a vector
|    Sun           Sun vector
|    Moon          Moon vector
|
|  References    :
|    None.
|
 ----------------------------------------------------------------------- }
{
```

```
}
PROCEDURE PertAccel          ( R,V                    : Vector;
                               ITime                  : Extended;
                               WhichOne               : Integer;
                               BC                     : Extended;
                               VAR APert              : Vector      );

   CONST
      OmegaEarth : Extended = 0.05883359068688786;
      TUDay      : Extended = 0.00933809102919444;
   VAR
      J2,J3,J4,gms,gmm,rs2,rm2,rs3,rm3,r32,r33,r34,r2,r3,r4,r5,r7,
      Beta,Temp,rho,srtasc,sdecl,mrtasc,mdecl         : EXTENDED;
      Va,RSun,RMoon         : VECTOR;
      i         : INTEGER;
   BEGIN
     MAG( R );
     MAG( V );

     R2:= r[4]*r[4];
     R3:= R2*r[4];
     R4:= R2*R2;
     R5:= R2*R3;
     R7:= R5*R2;
     R32:= r[3]*r[3];
     R33:= R32*r[3];
     R34:= R32*R32;

     CASE WhichOne OF
     { ------------------    J2 Acceleration    --------------------- }
       1 : BEGIN
             J2:=  0.00108263;
             APert[1]:= ( (-1.5*J2*r[1]) / R5 ) * ( 1.0-(5.0*R32) / R2 );
             APert[2]:= ( (-1.5*J2*r[2]) / R5 ) * ( 1.0-(5.0*R32) / R2 );
             APert[3]:= ( (-1.5*J2*r[3]) / R5 ) * ( 3.0-(5.0*R32) / R2 );
             MAG( APert );
           END;

     { ------------------    J3 Acceleration    --------------------- }
       2 : BEGIN
             J3:= -0.00000254;
             APert[1]:= ( (-2.5*J3*r[1]) / R7 ) * ((3.0*r[3])-(7.0*R33) / R2 );
             APert[2]:= ( (-2.5*J3*r[2]) / R7 ) * ((3.0*r[3])-(7.0*R33) / R2 );
             IF ABS( r[3] ) > 0.0000001 THEN
                 APert[3]:= ( (-2.5*J3*r[3]) / R7 ) * ((6.0*r[3])-((3.0*R33)
                                 / R2) - ((3.0*r2)  / r[3]))
               ELSE
                 APert[3]:= 0.0;
             MAG( APert );
           END;

     { ------------------    J4 Acceleration    --------------------- }
       3 : BEGIN
             J4:= -0.00000161;
             APert[1]:= ( (-1.875*J4*r[1]) / R7 ) * (1.0-((14.0*R32)/R2)+
                        ((21.0*R34) / R4 ));
             APert[2]:= ( (-1.875*J4*r[2]) / R7 ) * (1.0-((14.0*R32)/R2)+
                        ((21.0*R34) / R4 ));
             APert[3]:= ( (-1.875*J4*r[3]) / R7 ) * (5.0-((70.0*R32)/(3.0*R2))+
                        ((21.0*R34) / R4 ));
             MAG( APert );
           END;

     { ------------------    Sun Acceleration    --------------------- }
       4 : BEGIN
             GMS:=  3.329529364E05;
             ITime:= ITime * TUDay;
             SUN( ITime,RSun,SRtAsc,SDecl );
             FOR i:= 1 to 4 DO
                 RSun[i]:= RSun[i]*23455.07003;    { chg AU's to DU's }

             RS2:= RSun[4]*RSun[4];
             RS3:= RS2*RSun[4];
             APert[1]:= (-GMS/RS3) *
                        (r[1]-3.0*RSun[1]*
                              ((r[1]*RSun[1]+r[2]*RSun[2]+r[3]*RSun[3]) / RS2));
             APert[2]:= (-GMS/RS3) *
                        (r[2]-3.0*RSun[2]*
                              ((r[1]*RSun[1]+r[2]*RSun[2]+r[3]*RSun[3]) / RS2));
             APert[3]:= (-GMS/RS3) *
                        (r[3]-3.0*RSun[3]*
                              ((r[1]*RSun[1]+r[2]*RSun[2]+r[3]*RSun[3]) / RS2));
             MAG( APert );
           END;
{
```

```
}
    { ------------------ Moon Acceleration   --------------------- }
     5 : BEGIN
          GMM:=  0.01229997;
          ITime:= ITime * TUDay;
          MOON( ITime,RMoon,MRtAsc,MDecl );
          RM2:= RMoon[4]*RMoon[4];
          RM3:= RM2*RMoon[4];
          APert[1]:= (-GMM/RM3) *
                     (r[1]-3.0*RMoon[1]*
                           ((r[1]*RMoon[1]+r[1]*RMoon[1]+r[3]*RMoon[3]) / RM2));
          APert[2]:= (-GMM/RM3) *
                     (r[2]-3.0*RMoon[2]*
                           ((r[1]*RMoon[1]+r[2]*RMoon[2]+r[3]*RMoon[3]) / RM2));
          APert[3]:= (-GMM/RM3) *
                     (r[3]-3.0*RMoon[3]*
                           ((r[1]*RMoon[1]+r[3]*RMoon[3]+r[3]*RMoon[3]) / RM2));
          MAG( APert );
        END;

    { ------------------ Drag Acceleration   --------------------- }
     6 : BEGIN
          Va[1]:= V[1] + (OmegaEarth*r[2]);  { DU/TU }
          Va[2]:= V[2] - (OmegaEarth*r[1]);
          Va[3]:= V[3];
          MAG( Va );

          ATMOS( R, Rho );

          Temp:= -1000.0 * Va[4] * 0.5*Rho* ( 1.0/BC )* 6378137.0;
          APert[1]:= Temp*Va[1];
          APert[2]:= Temp*Va[2];
          APert[3]:= Temp*Va[3];
          MAG( APert );
        END;

    { ------------------ Solar Acceleration   --------------------- }
     7 : BEGIN
          ITime:= ITime * TUDay;
          SUN( ITime,RSun,SRtAsc,SDecl );
          FOR i:= 1 to 4 DO
              RSun[i]:= RSun[i]*23455.07003;   { chg AU's to DU's }

          Beta:= 0.4;
          APert[4]:= (4.74E-06*(1.0+Beta))/(BC*9.807);
          APert[1]:= (-APert[4]*RSun[1])/RSun[4];
          APert[2]:= (-APert[4]*RSun[2])/RSun[4];
          APert[3]:= (-APert[4]*RSun[3])/RSun[4];
        END;
    END;  { Case }

  END;  { Procedure PertAccel }
 {
```

```
}
{ ---------------------------------------------------------------------------
|
|                          PROCEDURE PDERIV
|
| This procedure calculates the derivative of the state vector for use with
|   the Runge-Kutta algorithm.  The DerivType string is used to determine
|   which perturbation equations are used.
|
| Algorithm     : Assign values
|                 Check each value of Derivtype and if a perturbation is needed
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
|
| Inputs        :
|   Time        - Time                                TU
|   X           - State Vector                        DU   ,  DU/TU
|   DerivType   - String containing YN for incl perts 'YYNYNYN'
|   BC          - Ballistic Coefficient               kg/m2
|
| Outputs       :
|   XDot        - Derivative of State Vector          DU\TU, DU\TU2
|
| Locals        :
|   RCubed      - Cube of X(4)
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
| Referances    :
|   None.
|
 --------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE PDeriv            ( X                    : Matrix;
                              ITime                : Extended;
                              DerivType            : Str10;
                              BC                   : Extended;
                              VAR XDot             : Matrix );
  VAR
    RCubed : Extended;
    Ro,Vo,APert,TempPert: Vector;
    i: INTEGER;
  BEGIN
   FOR i:= 1 to 3 DO
     BEGIN
       APert[i]:= 0.0;
       Ro[i] := GetVal(x,i,1);
       Vo[i] := GetVal(x,i+3,1);
     END;
   MAG( Ro );
   MAG( Vo );
   APert[4]:= 0.0;

   RCubed:= Ro[4]*Ro[4]*Ro[4];

   { -------------------- Velocity Terms ---------------------- }
   AssignVal( XDot,1,1, GetVal(X,4,1) );
   AssignVal( XDot,2,1, GetVal(X,5,1) );
   AssignVal( XDot,3,1, GetVal(X,6,1) );

   { ------------------ Acceleration Terms -------------------- }
   IF DerivType[1] = 'Y' THEN
       PertAccel( Ro,Vo,ITime,1,BC, APert );
   IF DerivType[2] = 'Y' THEN
     BEGIN
       PertAccel( Ro,Vo,ITime,2,BC, TempPert );
       AddVec( TempPert,APert,APert );
     END;
   IF DerivType[3] = 'Y' THEN
     BEGIN
       PertAccel( Ro,Vo,ITime,3,BC, TempPert );
       AddVec( TempPert,APert,APert );
     END;
   IF DerivType[4] = 'Y' THEN
     BEGIN
       PertAccel( Ro,Vo,ITime,4,BC, TempPert );
       AddVec( TempPert,APert,APert );
     END;
   IF DerivType[5] = 'Y' THEN
     BEGIN
       PertAccel( Ro,Vo,ITime,5,BC, TempPert );
       AddVec( TempPert,APert,APert );
     END;
   IF DerivType[6] = 'Y' THEN
     BEGIN
       PertAccel( Ro,Vo,ITime,6,BC, TempPert );
       AddVec( TempPert,APert,APert );
     END;
   IF DerivType[7] = 'Y' THEN
     BEGIN
       PertAccel( Ro,Vo,ITime,7,BC, TempPert );
       AddVec( TempPert,APert,APert );
     END;

    AssignVal( XDot,4,1,(-GetVal(X,1,1) / RCubed) + APert[1] );
    AssignVal( XDot,5,1,(-GetVal(X,2,1) / RCubed) + APert[2] );
    AssignVal( XDot,6,1,(-GetVal(X,3,1) / RCubed) + APert[3] );

  END;  { Procedure PDeriv }
{
```

```
}
{ ----------------------------------------------------------------------------
|
|                          PROCEDURE RK4
|
| This procedure is a fourth order Runge-Kutta integrator for an
|   N-dimensional First Order differential equation.  The user must provide
|   an external subroutine containing the system Equations of Motion.  Notice
|   time is included since some applications may need this.  The LAST position
|   in DerivType is a flag for two-body motion.  Two-Body motion is used if
|   the 10th element is set to '2', otherwise the Yes and No values determine
|   which perturbations to use.
|
| Algorithm     : Evaluate each term depending on the derivtype
|                 Find the final answer
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
| Inputs        :
|   ITime       - Initial Time                          TU
|   DT          - Step size                             TU
|   N           - Dimension of the state
|   DERIVTYPE   - Which perturbations to use
|   BC          - Ballistic Coefficient                 kg/m2
|   X           - State vector at initial time          DU, DU/TU
|
| Outputs       :
|   X           - State vector at new time              DU, DU/TU
|
| Locals        :
|   XDot        - Derivative of State Vector
|   Time        - Time                                  TU
|   K           - Storage
|   TEMP        - Storage
|   J           - Index
|   TempTime    - Tempoary time storage                 TU
|
| Constants     :
|   None.
|
| Coupling      :
|   Deriv         Procedure for Derivatives of E.O.M.
|
| References    :
|   James, et al, "Applied Num Methods"  pg. 461-466, eqtn pg. 463.
|   BMW           pg. 414-415
|
 --------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE RK4                    ( ITime                  : Extended;
                                   DT                     : Extended;
                                   N                      : Integer;
                                   DerivType              : Str10;
                                   BC                     : Extended;
                                   VAR X                  : Matrix );
  VAR
      XDot, Temp          : Matrix;
      Time,TempTime       : Extended;
      K                   : Matrix;
      J                   : Integer;
  BEGIN
    { ---------------------- Initialize X Dot --------------------- }
    InitMatrix( 10,3,K );
    InitMatrix( 6,1,Temp );
    InitMatrix( 6,1,XDot );

    IF DerivType[10] = '2' THEN
        DERIV( ITime,X, XDot )
      ELSE
        PDeriv( X,ITime,DerivType,BC, XDot );

    TempTime:= ITime;

    { -------------- Evaluate 1st Taylor Series Term -------------- }
    FOR j:= 1 to N DO
      BEGIN
        AssignVal( K,   J,1, Dt * GetVal( XDot,J,1) );
        AssignVal( Temp,J,1, GetVal( X,J,1) + 0.5*GetVal(K,J,1) );
      END;

    Time:= itime + Dt/2.0;

    IF DerivType[10] = '2' THEN
        DERIV( Time,Temp, XDot )
      ELSE
        PDeriv( Temp,Time,DerivType,BC,  XDot );

    { -------------- Evaluate 2nd Taylor Series Term -------------- }
    FOR j:= 1 to N DO
      BEGIN
        AssignVal( K,J,2,Dt * GetVal( XDot,J,1) );
        AssignVal( Temp,J,1,GetVal( X,J,1) + 0.5*GetVal(K,J,2) );
      END;
    IF DerivType[10] = '2' THEN
        DERIV( Time,Temp, XDot )
      ELSE
        PDeriv( Temp,Time,DerivType,BC,  XDot );

    { -------------- Evaluate 3rd Taylor Series Term -------------- }
    FOR j:= 1 to N DO
      BEGIN
        AssignVal( K,J,3,Dt * GetVal( XDot,J,1) );
        AssignVal( Temp,J,1, GetVal( X,J,1) + GetVal(K,J,3) );
      END;

    TempTime:= Temptime + Dt;

    IF DerivType[10] = '2' THEN
        DERIV( TempTime,Temp, XDot )
      ELSE
        PDeriv( Temp,TempTime,DerivType,BC,  XDot );

    { - ----------------- Update the State vector  --------------- }
    FOR j:= 1 to N DO
        AssignVal( X,J,1,GetVal( X,J,1) +
              ( GetVal(K,J,1) + 2.0*( GetVal(K,J,2)+GetVal(K,J,3) ) +
              Dt*GetVal( XDot,J,1) ) / 6.0 );

    DelMatrix( K );
    DelMatrix( Temp );
    DelMatrix( XDot );

  END;  { Procedure RK4 }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE ATMOS
|
|  This procedure finds the atmospheric density at an altitude above an
|    oblate earth given the position vector in the Geocentric Equatorial
|    frame.  The position vector is in DU's and the density is in gm/cm**3.
|
|  Algorithm     : Find initial values
|                  Loop to find the latitudes
|                  Calculate the density through a cascading IF statement
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
|
|  Inputs        :
|    R           - GEC Position vector                     DU
|
|  Outputs       :
|    Rho         - Density                                 kg/m**3
|
|  Locals        :
|    Rc          - Range of site w.r.t. earth center       DU
|    Height      - Height above earth w.r.t. site          DU
|    Alt         - Altitude above earth w.r.t. site        km
|    OldDelta    - Previous value of DeltaLat              rad
|    DeltaLat    - Diff between Delta and Geocentric lat  rad
|    GeoDtLat    - Geodetic Latitude                      -Pi/2 to Pi/2 rad
|    GeoCnLat    - Geocentric Latitude                    -Pi/2 to Pi/2 rad
|    TwoFMinusF2 - 2*F - F squared
|    OneMinusF2  - ( 1 - F ) squared
|    Delta       - Declination angle of R in IJK system    rad
|    Temp        - Diff between Geocentric/Geodetic lat    rad
|    RSqrd       - Magnitude squared
|    SinTemp     - Sine of Temp
|    RhoNom      - Nominal density at particular alt       gm/cm**3
|    H           - Scale Height                            km
|    i           - index
|
|  Constants     :
|    Pi          -                                         3.14159265358979
|    Flat        - Flatenning of the Earth                 0.003352810664747352
|    REarthKm    - Earth equatorial radius                 6378.137
|
|  Coupling      :
|    MAG           Magnitude of a vector
|
|  References    :
|    Escobal       pg. 398-399 ( Conversion to Lat and Height )
|
 ---------------------------------------------------------------------- }
{
```

```
}
PROCEDURE ATMOS                 ( R                        : Vector;
                                  VAR Rho                  : Extended );
   CONST
      Pi   : Extended = 3.14159265359;
      Flat: Extended = 0.003352810664747352;
   VAR
      Rc, Height, OldDelta, DeltaLat, GeoDtLat, TwoFMinusF2, SinTemp,
      OneMinusF2, Delta, RSqrd, Temp, GeoCnLat, h, Alt, RhoNom  : Extended;
      i              : Integer;
   BEGIN
     { --------------------  Initialize values   ------------------- }
     MAG( R );
     TwoFMinusF2:= 2.0*Flat - Flat*Flat;
     OneMinusF2  := POWER( 1.0-Flat,2.0 );

     { -------------- Set up initial latitude value  -------------- }
     Delta:= ArcTan( R[3] / SQRT( R[1]*R[1] + R[2]*R[2] ) );
     IF ABS( Delta ) > Pi THEN
         Delta:= RealMOD( Delta,Pi );
     GeoCnLat:= Delta;
     OldDelta:=  1.0;
     DeltaLat:= 10.0;
     RSqrd   := R[4]*R[4];

     { ----- Iterate to find Geocentric and Geodetic Latitude  ----- }
     i:= 1;
     WHILE ( ABS( OldDelta - DeltaLat ) > 0.00001 ) and ( i < 10 ) DO
       BEGIN
         OldDelta:= DeltaLat;
         Rc      := SQRT( ( 1.0-TwoFMinusF2 ) /
                          ( 1.0-TwoFMinusF2*COS(GeoCnLat)*COS(GeoCnLat) ) );
         GeoDtLat:= ArcTan( TAN(GeoCnLat) / OneMinusF2 );
         Temp    := GeoDtLat-GeoCnLat;
         SinTemp:= SIN( Temp );
         Height  := SQRT( RSqrd-Rc*Rc*SinTemp*SinTemp ) - Rc*COS(Temp);
         DeltaLat:= ARCSIN( Height*SinTemp / R[4] );
         GeoCnLat:= Delta - DeltaLat;
         INC( i );
       END;  { While }

     IF i >= 10 THEN
         WriteLn( 'IJKtoLatLon did NOT converge ' );
{
```

```
}
    ALT:= Height*6378.137;

    { ----------- Determine density based on altitude ------------ }
     IF Alt >= 800 THEN
      BEGIN
       H      := 130.8;
       RHONOM:=   4.262E-17;
       RHO    := RHONOM*EXP((800.0-ALT)/H);
      END
      ELSE
       IF Alt >= 700 THEN
        BEGIN
         H      := 105.3;
         RHONOM:=   1.216E-16;
         RHO    := RHONOM*EXP((700.0-ALT)/H);
        END
        ELSE
         IF Alt >= 600 THEN
          BEGIN
           H      := 91.0;
           RHONOM:=  3.818E-16;
           RHO    := RHONOM*EXP((600.0-ALT)/H);
          END
          ELSE
           IF Alt >= 500 THEN
            BEGIN
             H      := 81.9;
             RHONOM:=  1.316E-15;
             RHO    := RHONOM*EXP((500.0-ALT)/H);
            END
            ELSE
             IF Alt >= 400 THEN
              BEGIN
               H      := 73.2;
               RHONOM:=  5.192E-15;
               RHO    := RHONOM*EXP((400.0-ALT)/H);
              END
              ELSE
               IF Alt >= 300 THEN
                BEGIN
                  H      := 61.2;
                  RHONOM:=  2.653E-14;
                  RHO    := RHONOM*EXP((300.0-ALT)/H);
                END
                ELSE
                 IF Alt >= 250 THEN
                  BEGIN
                    H      := 52.6;
                    RHONOM:=  7.316E-14;
                    RHO    := RHONOM*EXP((250.0-ALT)/H);
                  END
                  ELSE
                   IF Alt >= 200 THEN
                    BEGIN
                      H      := 40.8;
                      RHONOM:=  2.706E-13;
                      RHO    := RHONOM*EXP((200.0-ALT)/H);
                    END
                    ELSE
                     IF Alt >= 150 THEN
                      BEGIN
                        H      := 24.1;
                        RHONOM:=  2.141E-12;
                        RHO    := RHONOM*EXP((150.0-ALT)/H);
                      END
                      ELSE
                       IF Alt >= 130 THEN
                        BEGIN
                          H      := 16.1;
                          RHONOM:=  8.484E-12;
                          RHO    := RHONOM*EXP((130.0-ALT)/H);
                        END
                        ELSE
                          BEGIN
                            H      :=  8.06;
                            RHONOM:=  9.661E-11;
                            RHO    := RHONOM*EXP((110.0-ALT)/H);
                          END;

   END;  { Procedure Atmos }
{
```

```
}
{ -------------------------------------------------------------------------
|
|                            PROCEDURE CHEBY
|
| This procedure calculates a CHEBYCHEV expansion for the atmosphere.
|   Given an altitude above the Earth's surface, it will find the pressure and
|   density at that altitude using a Chebyshev polynomial.  Calculations are
|   accomplished in metric units, and the final answers are converted to
|   English units, as described below.
|   The model is only valid from 0 to 200 km (656,000 ft) altitude.
|
| Algorithm     : Convert the altitude to km
|                 Assign the pressure coeff based on altitude
|                 Calculate the pressure
|                 Assign the density coeff based on altitude
|                 Calculate the denisty
|                 Convert to ENGLISH units
|
| Author        : C2C Gandhi         USAFA       719-472-4109  28 Nov 1988
|                 Capt Dave Vallado  USAFA/DFAS  719-472-4109  28 Aug 1990
|
| Inputs        :
|   Alt         - Altitude above earth's surface,        ft
|
| Outputs       :
|   PAlt        - Pressure at altitude                   lbf/in**2
|   RhoAlt      - Density at altitude                    lbm/ft**3
|
| Locals        :
|   ..          - ..                                     .
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
| References    :
|   None.
|
 ------------------------------------------------------------------------- }

PROCEDURE CHEBY                ( ALT                      : Extended;
                                 VAR PALT,RHOALT          : Extended );
   VAR
      Z, Z1, Sum, P0 , X, Nu, Part, LnR, R, Rho0 : Extended;
      C,A : ARRAY[1..15] of Extended;
      k          : INTEGER;
   BEGIN
     { --------------- Convert altitude to kilometers -------------- }
      Z := Alt * 0.0003048;
{
```

```
}
  IF Z <= 80.0 THEN
    BEGIN
  { ------- Chebychev model for altitudes of 80 km or less ------ }
  { ----- Define initial and zero altitude pressure constants --- }
      SUM   :=       0.0;
      Z1    :=      80.0;
      P0    := 101325.0;
  { --------- Define the pressure ratio coefficients ----------- }
      A[1]  := -11.385925;
      A[2]  :=  -5.6837011;
      A[3]  :=   0.052666476;
      A[4]  :=  -0.077884294;
      A[5]  :=  -0.11004083;
      A[6]  :=   0.017572339;
      A[7]  :=   0.0048546337;
      A[8]  :=   0.0017694805;
      A[9]  :=  -0.0018185298;
      A[10] :=  -0.0026635086;
      A[11] :=   0.0035685433;
      A[12] :=  -0.00082257517;
      A[13] :=  -0.0010363683;
      A[14] :=   0.00057053477;
      A[15] :=  -0.00019023078;
    END
    ELSE
    BEGIN
  { ------- Chebychev model for altitudes of 80 to 200 km ------- }
  { --- Define initial and zero altitude pressure constants ----- }
      SUM :=      0.0;
      Z1  :=    200.0;
      P0  := 101325.0;
  { ------------- Define the pressure ratio coefficients -------- }
      A[1]  := -24.475069;
      A[2]  := -10.685861;
      A[3]  :=  2.2622605;
      A[4]  :=  0.63433398;
      A[5]  := -0.27948959;
      A[6]  := -0.31548574;
      A[7]  :=  0.090751361;
      A[8]  :=  0.18530467;
      A[9]  := -0.095325843;
      A[10] := -0.050214309;
      A[11] :=  0.045101378;
      A[12] :=  0.0088997472;
      A[13] := -0.018935899;
      A[14] :=  0.0035690621;
      A[15] := -0.0063989880;
    END;

  { ------- Define X as a function of the altitude ratio -------- }
   X     := 2.0 * Z/Z1 - 1.0;

  { --------------- Define Nu as a function of X --------------- }
   Nu    := 2.0 * X;

  { ----- Define the Chebyshev Polynomials as functions of Nu --- }
   C[2]  := Nu;
   C[3]  := Nu*Nu - 2.0;
   FOR k:= 4 to 15 DO
       C[k]  := Nu * C[k-1] - C[k-2];

  { - Sum all parts of the Chebyshev expansion atmosph·ric model- }
   FOR k:= 2 to 15 DO
     BEGIN
       PART := A[k] * C[k];
       SUM  := SUM + PART;
     END;

  { -------------- Solve for the pressure at altitude ----------- }
   LNR  := 0.5 * (A[1] + SUM);
   R    := EXP(LNR);
   PALT := R * P0;
{
```

```
}
      IF Z <= 80.0 THEN
        BEGIN
    { ------- Chebychev model for altitudes of 80 km or less ------ }
    { ------- Define initial and zero altitude density constants -- }
          SUM   :=  0.0;
          Z1    := 80.0;
          RHO0  :=  1.2250;
    { ------------ Define the density ratio coefficients ---------- }
          A[1]  := -10.960632;
          A[2]  := -5.5717132;
          A[3]  :=  0.099116555;
          A[4]  :=  0.061044847;
          A[5]  := -0.14304157;
          A[6]  :=  0.0029494088;
          A[7]  :=  0.0058789604;
          A[8]  :=  0.0020421324;
          A[9]  :=  0.0071033206;
          A[10] := -0.0010314086;
          A[11] :=  0.0034100737;
          A[12] :=  0.0041764325;
          A[13] := -0.0039151559;
          A[14] :=  0.0011227828;
          A[15] := -0.0015751053;
        END
        ELSE
        BEGIN
    { ------- Define initial and zero altitude density constants -- }
          SUM := 0.0;
          Z1  := 200.0;
          RHO0  := 1.2250;
    { ------- Chebychev model for altitudes of 80 to 200 km ------- }
    { ------------ Define the density ratio coefficients ---------- }
          A[1]  := -25.415229;
          A[2]  := -11.684380;
          A[3]  :=  1.8721406;
          A[4]  :=  0.81660876;
          A[5]  := -0.093811118;
          A[6]  := -0.30155735;
          A[7]  := -0.077593291;
          A[8]  :=  0.21640168;
          A[9]  := -0.034918422;
          A[10] := -0.070126799;
          A[11] :=  0.036014616;
          A[12] :=  0.014951351;
          A[13] := -0.021450283;
          A[14] := -0.0012497995;
          A[15] :=  0.018421866;
        END;

    { --------- Define X as a function of the altitude ratio ------ }
     X      := 2.0 * Z/Z1 - 1.0;

    { ------------------- Define Nu as a function of X ------------ }
     Nu     := 2.0 * X;

    { ------ Define the Chebyshev Polynomials as functions of Nu -- }
     C[2]   := Nu;
     C[3]   := Nu*Nu - 2.0;
     FOR k:=4 to 15 DO
         C[k]  := Nu * C[k-1] - C[k-2];

    { - Sum all parts of the Chebyshev expansion atmospheric model  }
     FOR k:= 2 to 15 DO
       BEGIN
         PART := A[k] * C[k];
         SUM  := SUM + PART;
       END;

    { -------------- Solve for the density at altitude ------------ }
     LNR  := 0.5 * (A[1] + SUM);
     R    := EXP(LNR);
     RHOALT := R * RHO0;

    { --- Convert pressure & density from metric units to English units }
    { ---     (N/(m*m) ==> lbf/in**2;  kg/m**3 ==> lbm/ft**3) -------- }
     PAlt   := PAlt * 0.000145;
     RhoAlt := RhoAlt * 0.062429507;

  END;  { Procedure Cheby }
{
```

APPENDIX B

PASCAL SOURCE CODE

MATHEMATICAL ROUTINES

```
UNIT MATH;

(*     --------------------------------------------------------------        *)
(*                                                                           *)
(*                          Module - MATH.PAS                                *)
(*                                                                           *)
(*  This file contains most of the math procedures and functions.            *)
(*                                                                           *)
(*     ****************  NOTICE OF GOVERNMENT ORIGIN  ****************       *)
(*                                                                           *)
(*  This software has been developed by an employee of the United States     *)
(*  Government at the United States Air Force Academy, and is therefore      *)
(*  a work of the United States, and is NOT subject to copyright protection  *)
(*  under the provisions of 17 U.S.C. 105.  ANY use of this work, or         *)
(*  inclusion in other works, must comply with the notice provisions of      *)
(*  17 U.S.C. 403.                                                           *)
(*                                                                           *)
(*     ***************************************************************       *)
(*                                                                           *)
(*  Current : 30 Jan 91 Capt Dave Vallado        VERSION 3.0                 *)
(*                                                                           *)
(*  Changes : 25 Jan 91 Capt Dave Vallado                                    *)
(*                      Update formatting and misc fixes                     *)
(*            20 Sep 90 Capt Dave Vallado                                    *)
(*                      Add roots solvers and fix small                      *)
(*            20 Apr 90 Capt Dave Vallado        VERSION 2.0                 *)
(*                      Updated version                                      *)
(*            24 Jan 90 Capt Dave Vallado                                    *)
(*                      Change Matrix structure                              *)
(*             4 Dec 89 Capt Dave Vallado                                    *)
(*                      Added LOG function                                   *)
(*             8 Sep 88 Capt Dave Vallado                                    *)
(*                      Version 1.0                                          *)
(*            27 Jun 88 Capt Dave Vallado                                    *)
(*                      Fixes to MAG calls in misc procedures                *)
(*            17 Apr 88 Capt Dave Vallado                                    *)
(*                      Upgrade to Turbo 4.0                                 *)
(*            23 Nov 87 Capt Dave Vallado                                    *)
(*                      Incorporated comments for each procedure             *)
(*                                                                           *)
(*     --------------------------------------------------------------        *)

                              INTERFACE

(*     --------------------------------------------------------------        *)

   TYPE
       Vector    = ARRAY[1..4] of Extended;
       Str64     = STRING[64];

       Intarray = ARRAY[1..6] of Integer;

       MatrixDataPtr = ^MatrixData;
       MatrixData    = RECORD
                           Index      : Integer;
                           Number     : Extended;
                           Next,Last  : MatrixDataPtr;
                         END;
       MatrixPtr     = RECORD
                           NumRows,NumCols : Integer;
                           DPtr            : MatrixDataPtr;
                           Head,Tail       : MatrixDataPtr;
                         END;
       Matrix        = ^MatrixPtr;

    { ------------------- Misc functions for PASCAL -------------- }

  Function  SGN             ( XVal                      : Extended ) : Extended;

  Function  RealMOD         ( XVal, Modby               : Extended ) : Extended;

  Function  Power           ( Base, Pwr                 : Extended ) : Extended;

  Function  Log             ( XVal                      : Extended ) : Extended;

  Function  Min             ( X, Y                      : Extended ) : Extended;

  Function  Max             ( X, Y                      : Extended ) : Extended;

  Procedure Plane           ( x1,y1,z1,x2,y2,z2,x3,y3,z3: Extended;
                              VAR a,b,c,d               : Extended );
{
```

```
}
  { ------------------- Trigonometric Functions --------------- }

Function  Tan        ( XVal                    : Extended ) : Extended;
Function  Cot        ( XVal                    : Extended ) : Extended;
Function  Csc        ( XVal                    : Extended ) : Extended;
Function  Sec        ( XVal                    : Extended ) : Extended;
Function  ATan2      ( SinValue, CosValue      : Extended ) : Extended;
Function  ArcSin     ( XVal                    : Extended ) : Extended;
Function  ArcCos     ( XVal                    : Extended ) : Extended;
Function  Cosh       ( XVal                    : Extended ) : Extended;
Function  ArcCosh    ( XVal                    : Extended ) : Extended;
Function  Sinh       ( XVal                    : Extended ) : Extended;
Function  ArcSinh    ( XVal                    : Extended ) : Extended;
Function  Tanh       ( XVal                    : Extended ) : Extended;
Function  ArcTanh    ( XVal                    : Extended ) : Extended;

  { ------------------------- Vector Operations ----------------- }

Function  DOT        ( Vec1,Vec2               : Vector   ) : Extended;
Procedure CROSS      ( Vec1,Vec2               : Vector;
                       VAR OutVec              : Vector   );
Procedure MAG        ( VAR Vec                 : Vector   );
Procedure NORM       ( Vec                     : Vector;
                       VAR OutVec              : Vector   );
Procedure ROT1       ( Vec                     : Vector;
                       XVal                    : Extended;
                       VAR OutVec              : Vector   );
Procedure ROT2       ( Vec                     : Vector;
                       XVal                    : Extended;
                       VAR OutVec              : Vector   );
Procedure ROT3       ( Vec                     : Vector;
                       XVal                    : Extended;
                       VAR OutVec              : Vector   );
Procedure ADDVEC     ( Vec1,Vec2               : Vector;
                       VAR OutVec              : Vector   );
Procedure ADD3VEC    ( Vec1,Vec2,Vec3          : Vector;
                       VAR OutVec              : Vector   );
Procedure LNCOM1     ( A                       : Extended;
                       Vec                     : Vector;
                       VAR OutVec              : Vector   );
Procedure LNCOM2     ( A1,A2                   : Extended;
                       Vec1,Vec2               : Vector;
                       VAR OutVec              : Vector   );
Procedure LNCOM3     ( A1,A2,A3                : Extended;
                       Vec1,Vec2,Vec3          : Vector;
                       VAR OutVec              : Vector   );
Procedure ANGLE      ( Vec1,Vec2               : Vector;
                       VAR Theta               : Extended );

{
```

```
}
   { ------------------------- Analytic routines ----------------- }

  Procedure Quadratic          ( a,b,c                    : Extended;
                                 VAR R1r,R1i,R2r,R2i      : Extended  );

  Procedure Cubic              ( a,b,c,d                   : Extended;
                                 VAR R1r,R1i,R2r,R2i,R3r,R3i: Extended  );

  Procedure Quartic            ( a,b,c,d,e                 : Extended;
                                 VAR R1r,R1i,R2r,R2i,R3r,R3i,
                                     R4r,R4i               : Extended  );

   { ------------------------- Matrix Operations ----------------- }

  Procedure InitMatrix         ( Rows,Cols                : Integer;
                                 VAR A                    : Matrix      );

  Procedure DelMatrix          ( VAR A                    : Matrix      );

  Function  GetVal             ( VAR A                    : Matrix;
                                 Row,Col                  : Integer     ): Extended;

  Procedure AssignVal          ( VAR A                    : Matrix;
                                 Row,Col                  : Integer;
                                 Number                   : Extended    );

  Procedure MatMult            ( Mat1,Mat2                : Matrix;
                                 Mat1r,Mat1c,Mat2c        : Integer ;
                                 VAR Mat3                 : Matrix      );

  Procedure MatAdd             ( Mat1,Mat2                : Matrix;
                                 Mat1r,Mat1c              : Integer ;
                                 VAR Mat3                 : Matrix      );

  Procedure MatTrans           ( Mat1                     : Matrix;
                                 Mat1r,Mat1c              : Integer ;
                                 VAR Mat2                 : Matrix      );

  Procedure LUDeComp           ( VAR LU                   : Matrix;
                                 VAR Index                : Intarray;
                                 Order                    : Integer     );

  Procedure LUBkSub            ( LU                       : Matrix;
                                 Index                    : Intarray;
                                 Order                    : Integer;
                                 Var B                    : Matrix      );

  Procedure MatInverse         ( Mat                      : Matrix;
                                 Order                    : Integer;
                                 VAR MatInv               : Matrix      );

  Procedure PrintMat           ( Mat1                     : Matrix;
                                 Title                    : Str64       );

  Function  Determinant        ( Mat1                     : Matrix;
                                 Order                    : Integer     ) : Extended;

(*      -----------------------------------------------------------------      *)

                              IMPLEMENTATION

(*      -----------------------------------------------------------------      *)
{
```

```
}
{ -----------------------------------------------------------------------------
|
|                               FUNCTION SGN
|
|  This Function determines the sign of a number.
|
|  Algorithm     : Set the function to 1.0 if positive, -1.0 if negative
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   18 Sep 1990
|
|  Inputs        :
|    XVal        - Value to determine sign of
|
|  OutPuts       :
|    SGN         - Result                                       +1 or -1
|
|  Locals        :
|    None.
|
|  Coupling      :
|    None.
|
 ----------------------------------------------------------------------------- }

FUNCTION SGN               ( XVal                    : Extended ) : Extended;
   VAR
     Temp : EXTENDED;
   BEGIN
     IF XVal > 0.0 THEN
         Temp:= 1.0
       ELSE
         Temp:= -1.0;
     SGN:= Temp;
   END;  { Function SGN }


{ -----------------------------------------------------------------------------
|
|                               FUNCTION RealMOD
|
|  This Function performs the MOD operation for REALs.
|
|  Algorithm     : Assign a temporary variable
|                  Subtract off an integer number of values while the xval is
|                     too large
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
|  Inputs        :
|    XVal        - Value to MOD
|    ModBy       - Value to MOD with
|
|  OutPuts       :
|    RealMOD     - Result                   -ModBy <=  Answer  <= +ModBy
|
|  Locals        :
|    TempValue   - Temporary Extended value
|
|  Coupling      :
|    None.
|
 ----------------------------------------------------------------------------- }

FUNCTION RealMOD           ( XVal,Modby              : Extended ) : Extended;
   VAR
       TempValue: Extended;
   BEGIN
     TempValue := XVal;
     WHILE ABS(TempValue) > ModBy DO
         TempValue:= TempValue - INT(XVal/ModBy)*ModBy;
     RealMOD:= TempValue;
   END;  { Function RealMOD }

{
```

```
}
{ ---------------------------------------------------------------------------
|
|                              FUNCTION POWER
|
| This Function performs the raising of a base to a power.  Notice the IF
|   statement to eliminate problems such as a negative base, or a base equal
|   to 0.0.
|
| Algorithm      : If the base is positive, calculate the answer
|                  Otherwise, write an error
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs         :
|   Base         - Base value
|   Pwr          - Power to raise base to
|
| OutPuts        :
|   Power        - Result
|
| Locals         :
|   None.
|
| Coupling       :
|   None.
|
 --------------------------------------------------------------------------- }

FUNCTION Power                ( Base, Pwr                 : Extended ) : Extended;
   BEGIN
     IF Base > 0.0 THEN
         Power := Exp( Pwr * Ln( Base ) )
       ELSE
       BEGIN
         Writeln( 'Error in Power with base = ',base,' and Pwr = ',Pwr );
       END;
   END;  { Function Power }


{ ---------------------------------------------------------------------------
|
|                              FUNCTION LOG
|
| This Function performs the LOG base 10 problem.
|
| Algorithm      : If the x is positive, calculate the answer
|                  Otherwise, set the answer to 0.0
|
| Author         : Maj Tom Riggs      USAFA/DFAS  719-472-4109    4 Dec 1989
|
| Inputs         :
|   X            - Value to take the Log base 10 of
|
| OutPuts        :
|   Log          - Result
|
| Locals         :
|   None.
|
| Coupling       :
|   None.
|
 --------------------------------------------------------------------------- }

FUNCTION LOG                  ( XVal                      : Extended ) : Extended;
   Const
      Lfac       : Extended = 0.4342944819;      {1.0/ln(10)}
   BEGIN
      IF XVal > 0.0 THEN
          Log := Lfac*ln(XVal)
        ELSE
          Log := -1.0e37;
   END;   { Function Log }

{
```

```
}
{ ---------------------------------------------------------------------------
|
|                            FUNCTION MIN
|
| This Function determines the minimum of 2 values.
|
| Algorithm      : If the x is less than y, set min to x
|                  otherwise, set min to y
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs         :
|   X            - Value number 1
|   Y            - Value number 2
|
| OutPuts        :
|   Min          - Minimum of x or y
|
| Locals         :
|   None.
|
| Coupling       :
|   None.
|
 --------------------------------------------------------------------------- }

FUNCTION Min                ( X, Y                        : Extended ) : Extended;
    BEGIN
      IF X < Y THEN
          Min := X
        ELSE
          Min := Y;
    END;  { Function Min }


{ ---------------------------------------------------------------------------
|
|                            FUNCTION MAX
|
| This Function determines the maximum of 2 values.
|
| Algorithm      : If the x is more than y, set max to x
|                  otherwise, set max to y
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs         :
|   X            - Value number 1
|   Y            - Value number 2
|
| OutPuts        :
|   Max          - Minimum of x or y
|
| Locals         :
|   None.
|
| Coupling       :
|   None.
|
 --------------------------------------------------------------------------- }

FUNCTION Max                ( X, Y                        : Extended ) : Extended;
    BEGIN
      IF X > Y THEN
          Max := X
        ELSE
          Max := Y;
    END;  { Function Max }

{
```

```
}
{ -------------------------------------------------------------------------
|
|                          PROCEDURE PLANE
|
| This procedure calculates the equation of a plane given 3 points
|   pt1 - x1,y1,z1, pt2 - x2,y2,z2, pt3 - x3,y3,z3 , and outputs the
|   a b c d variables describing the plane. NOTE that the general equation
|   of a plane is defined here as:  ax + by + cz + d = 0  and the values
|   are obtained by solving the ordered determinant  x  y  z   1
|                                                    x1 y1 z1  1   =0
|                                                    x2 y2 z2  1
|                                                    x3 y3 z3  1
|
|  Algorithm     : find the line differences for each set of points
|                  Calculate the coefficients of the plane
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
|  Inputs        :
|    x1,y1,z1    - point # 1
|    x2,y2,z2    - point # 2
|    x3,y3,z3    - point # 3
|
|  OutPuts       :
|    a,b,c,d     - constants for the equation of the plane
|
|  Locals        :
|    z23
|
|  Coupling      :
|    None.
|
 ------------------------------------------------------------------------- }

PROCEDURE Plane               ( x1,y1,z1,x2,y2,z2,x3,y3,z3: Extended;
                                VAR a,b,c,d               : Extended);
   VAR
     z23,y23,x23,yz23,yz32,xz23,xz32,xy23,xy32 : Extended;
   BEGIN
     z23:= z2-z3;
     y23:= y2-y3;
     x23:= x2-x3;

     yz23:= y2*z3;
     yz32:= y3*z2;
     xz23:= x2*z3;
     xz32:= x3*z2;
     xy23:= x2*y3;
     xy32:= x3*y2;

     a:= y1*z23 - z1*y23 + yz23 - yz32;
     b:= x1*z23 - z1*x23 + xz23 - xz32;
     c:= x1*y23 - y1*x23 + xy23 - xy32;
     d:= x1*(yz23 - yz32) - y1*(xz23-xz32) + z1*(xy23 -xy32);
   END;  { Procedure Plane }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                            FUNCTION TAN
|
| This Function finds the tangent of an angle in radians.
|
| Algorithm     : If the absolute value of XVal is zero, set tan to infinity
|                 otherwise, find the answer using the sine
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
| Inputs        :
|   XVal        - Angle to take Tangent of                    rad
|
| OutPuts       :
|   Tan         - Result
|
| Locals        :
|   Temp        - Temporary Real variable
|
| Constants     :
|   Inifinity   - Value to represent infinity
|   Small       - Tolerance value
|
 ------------------------------------------------------------------------ }

FUNCTION Tan                 ( XVal                    : Extended ) : Extended;
    CONST
       Infinity : Extended = 999999.9;
       Small    : Extended = 0.000001;
    VAR
       Temp : EXTENDED;
    BEGIN
      Temp := Cos( XVal );

      IF ABS( Temp ) < Small THEN
          Tan := Infinity
        ELSE
          Tan := Sin( XVal ) / Temp;
    END;  { Function Tan }

{ ------------------------------------------------------------------------
|
|                            FUNCTION COT
|
| This Function finds the Cotangent of an angle in radians.
|
| Algorithm     : If the absolute value of XVal is zero, set cot to infinity
|                 otherwise, find the answer using the tangent
|
| Author        :  Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
| Inputs        :
|   XVal        - Angle to take Cotangent of                  rad
|
| OutPuts       :
|   Cot         - Result
|
| Locals        :
|   Temp        - Temporary Real variable
|
| Constants     :
|   Inifinity   - Value to represent infinity
|   Small       - Tolerance value
|
 ------------------------------------------------------------------------ }

FUNCTION Cot                 ( XVal                    : Extended ) : Extended;
    CONST
       Infinity : Extended = 999999.9;
       Small    : Extended = 0.000001;
    VAR
       Temp : EXTENDED;
    BEGIN
      Temp:= Tan( XVal );

      IF ABS( Temp ) < Small THEN
          Cot := Infinity
        ELSE
          Cot := 1.0 / Temp;
    END;  { Function Cot }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                            FUNCTION CSC
|
| This Function finds the Cosecant of an angle in radians.
|
| Algorithm     : If the absolute value of XVal is zero, set csc to infinity
|                 otherwise, find the answer using the sine
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
| Inputs        :
|   XVal        - Angle to take Cosecant of                    rad
|
| OutPuts       :
|   Csc         - Result
|
| Locals        :
|   Temp        - Temporary Real variable
|
| Constants     :
|   Inifinity   - Value to represent infinity
|   Small       - Tolerance value
|
 --------------------------------------------------------------------------- }

FUNCTION Csc                 ( XVal                      : Extended ) : Extended;
    CONST
       Infinity : Extended = 999999.9;
       Small    : Extended = 0.000001;
    VAR
       Temp : EXTENDED;
    BEGIN
      Temp:= Sin( XVal );

      IF ABS( Temp ) < Small THEN
          Csc := Infinity
        ELSE
          Csc := 1.0 / Temp;
    END;  { Function Csc }

{ ---------------------------------------------------------------------------
|
|                            FUNCTION SEC
|
| This Function finds the secant of an angle in radians.
|
| Algorithm     : If the absolute value of XVal is zero, set sec to infinity
|                 otherwise, find the answer using the Cosine
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
| Inputs        :
|   XVal        - Angle to take secant of                      rad
|
| OutPuts       :
|   Sec         - Result
|
| Locals        :
|   Temp        - Temporary Real variable
|
| Constants     :
|   Inifinity   - Value to represent infinity
|   Small       - Tolerance value
|
 --------------------------------------------------------------------------- }

FUNCTION Sec                 ( XVal                      : Extended ) : Extended;
    CONST
       Infinity : Extended = 999999.9;
       Small    : Extended = 0.000001;
    VAR
       Temp : EXTENDED;
    BEGIN
      Temp    := Cos( XVal );

      IF ABS( Temp ) < Small THEN
          Sec := Infinity
        ELSE
          Sec := 1.0 / Temp;
    END;  { Function Sec }
{
```

```
}
{ -----------------------------------------------------------------------------
|
|                            FUNCTION ATAN2
|
|  This Function performs the arc tangent 2 function which resolves
|    quadrants.  The arguments passed are the sine and cosine values.
|
|  Algorithm     : Determine the quadrant using IF statments
|                  If the answer is not a sepcial case, 0, 180, etc
|                     find the arctangent
|                  otherwise, find the special case values
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
|  Inputs        :
|    SinValue    - Sine of desired angle                        rad
|    CosValue    - Cosine of desired value                      rad
|
|  OutPuts       :
|    Atan2       - Arctangent with resolved quadrants           0.0 to 2Pi rad
|
|  Locals        :
|    TanArg      - Temporary Extended Value
|    Quadrant    - Quadrant of the answer                       1 2 3 4
|    SinInteger  - Sign of the value                            +1 or -1
|
|  Constants     :
|    Pi            3.14159265359
|    TwoPi         6.28318530717959
|
|  Coupling      :
|    None.
|
 ---------------------------------------------------------------------------- }

FUNCTION ATan2                ( SinValue,CosValue         : Extended ) : Extended;
   CONST
     Pi    : Extended =  3.14159265359;
     TwoPi: Extended =  6.28318530717959;
   VAR
     TanArg                : Extended;
     Quadrant, SinInteger : Integer;
   BEGIN
     Quadrant:= 5;
     IF (SinValue > 0.0 ) and (SinValue < 1.0 ) and
        (CosValue > 0.0 ) and (CosValue < 1.0 ) THEN
          quadrant:= 1;
     IF (SinValue > 0.0 ) and (SinValue < 1.0 ) and
        (CosValue < 0.0 ) and (CosValue >-1.0 ) THEN
          quadrant:= 2;
     IF (SinValue > -1.0 ) and (SinValue < 0.0 ) and
        (CosValue <  0.0 ) and (CosValue >-1.0 ) THEN
          quadrant:= 3;
     IF (SinValue > -1.0 ) and (SinValue < 0.0 ) and
        (CosValue >  0.0 ) and (CosValue < 1.0 ) THEN
          quadrant:= 4;
     IF Quadrant <> 5 THEN
       BEGIN
         tanarg:= arctan(SinValue/CosValue);
         IF (Quadrant <  4) and (Quadrant <> 1) THEN
             tanarg:= tanarg + Pi
           ELSE
             IF Quadrant =  4 THEN
                 tanarg:= tanarg + TwoPi;
       END
       ELSE
         BEGIN                              { Special Cases }
           SinInteger:= Round(SinValue);
           CASE SinInteger OF
             -1 :  TanArg:= 3.0*Pi/2.0;
              0 :  IF ROUND(CosValue) > 0.0 THEN
                       TanArg:= 0.0
                     ELSE
                       TanArg:= Pi;
              1 :  TanArg:=  Pi/2.0;
           END; { Case }
         END;
     ATan2:= tanarg;
   END; { Function Atan2 }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                            FUNCTION ARCSIN
|
| This function evaluates Arc Sine using the standard Arc Tangent function.
|
| Author       : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Algorithm    : If the absolute value of the argument (XVal) is less than 1.0
|                    find the answer using the ARCTAN function
|                  else
|                If the absolute value of the argument (XVal) is very close to 1.0
|                  If XVal is positive
|                      the answer = 90 degrees
|                    else
|                      the answer = -90 degrees
|                   else
|                     write an error message to the screen
|
| Inputs       :
|   XVal       - Angle value                                -1.0 to 1.0 rad
|
| OutPuts      :
|   ArcSin     - Result                                     -Pi/2 to Pi/2 rad
|
| Locals       :
|   Temp       - Temporary Extended Value
|
| Constants    :
|   Pi         - Pi
|   Small      - Tolerance
|
| Coupling     :
|   None.
|
 -------------------------------------------------------------------------- }

FUNCTION ArcSin              ( XVal                      : Extended ) : Extended;
   CONST
       Small : Extended = 0.000001;
       Pi    : Extended = 3.14159265359;
   VAR
       Temp : Extended;
   BEGIN
     IF ABS( XVal ) < 1.0 THEN
         Temp:= ArcTan( XVal / SQRT(1.0 - XVal*XVal) )
       ELSE
         IF ABS(XVal)-1.0 < Small THEN
             IF XVal > 0.0  THEN
                 Temp:=  Pi / 2.0  { XVal =  1.0 }
               ELSE
                 Temp:= -Pi / 2.0  { XVal = -1.0 }
           ELSE
             Writeln( 'Error in ArcSin argument = ',XVal );

     ArcSin:= Temp;
   END;  { Function ArcSin }
{
```

```
}
{ -----------------------------------------------------------------------------
|
|                               FUNCTION ARCCOS
|
| This function evaluates Arc Cosine using the ArcSin function.
|
| Algorithm     : If the absolute value of the argument (XVal) is less than 1.0
|                     find the answer using the ARCTAN function
|                   else
|                     If the absolute value of the argument (XVal) is very close to 1.0
|                         If XVal is positive
|                             the answer = 90 degrees
|                         else
|                             the answer = -90 degrees
|                     else
|                       write an error message to the screen
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs        :
|   XVal        - Angle Value                                  -1.0 to 1.0 rad
|
| OutPuts       :
|   ArcCos      - Result                                       0.0 to Pi rad
|
| Locals        :
|   Temp        - Temporary Extended Value
|
| Constants     :
|   Pi          - Pi
|   Small       - Tolerance
|
| Coupling      :
|   ARCSIN        Sine of an angle in radians
|
 ---------------------------------------------------------------------------- }

FUNCTION ArcCos               ( XVal                    : Extended ) : Extended;
   CONST
       Small : Extended = 0.000001;
       Pi    : Extended = 3.14159265359;
   VAR
       Temp : Extended;
   BEGIN
     IF ABS(XVal) < 1.0 THEN
       BEGIN
         Temp:= ArcSin( SQRT(1.0 - XVal*XVal) );
         IF XVal < 0.0 THEN
             Temp:= Pi - Temp;
       END
       ELSE
         IF ABS(XVal)-1.0 < Small THEN
             IF XVal > 0.0  THEN
                 Temp:=  0.0    { XVal =  1.0 }
               ELSE
                 Temp:=  Pi     { XVal = -1.0 }
           ELSE
             Writeln( 'Error in ArcCos argument = ',XVal );

     ArcCos:= Temp;
   END;  { Function ArcCos }
{
```

```
}
{ -----------------------------------------------------------------------------
|
|                              FUNCTION COSH
|
|  This function evaluates the hyperbolic cosine function.
|
|  Algorithm     : Calculate the answer
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
|  Inputs        :
|    XVal        - Angle value                                 any real
|
|  OutPuts       :
|    Cosh        - Result                                      1.0 to Infinity
|
|  Locals        :
|    None.
|
|  Coupling      :
|    None.
|
 ----------------------------------------------------------------------------- }

FUNCTION Cosh                ( XVal                  : Extended ) : Extended;
   BEGIN
     Cosh:= 0.5*( EXP(XVal) + EXP(-XVal) );
   END;  { Function Cosh }


{ -----------------------------------------------------------------------------
|
|                              FUNCTION ARCCOSH
|
|  This function evaluates the inverse hyperbolic cosine function.
|
   Algorithm     : If XVal squared - 1 is less than zero, set to undefined
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
|  Inputs        :
|    XVal        - Angle Value                                 1.0 to Infinity
|
|  OutPuts       :
|    ArcCosh     - Result                                      any real
|
|  Locals        :
|    Temp        - Temporary Extended Value
|
|  Constants     :
|    Undefined   - Flag value for an undefined quantity
|
|  Coupling      :
|    None.
|
 ----------------------------------------------------------------------------- }

FUNCTION ARCCosh             ( XVal                  : Extended ) : Extended;
   CONST
       Undefined : Extended = 999999.1;
   VAR
       Temp : Extended;
   BEGIN
     IF XVal*XVal - 1.0 < 0.0 THEN
       BEGIN
         Temp:= Undefined;
         WriteLn( 'Error in ArcCosh Function ' );
       END
       ELSE
         Temp:= LN( XVal + SQRT( XVal*XVal - 1.0 ) );

     ArcCosh:= Temp;
   END;  { Function ArcCosh }
{
```

```
}
{ ------------------------------------------------------------------------------
|
|                              FUNCTION SINH
|
|  This function evaluates the hyperbolic sine function.
|
|  Algorithm     : Calculate the answer
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
|  Inputs        :
|    XVal        - Angle Value                                   any real
|
|  OutPuts       :
|    Sinh        - Result                                        any real
|
|  Locals        :
|    None.
|
|  Coupling      :
|    None.
|
 ------------------------------------------------------------------------------ }

FUNCTION Sinh                 ( XVal                  : Extended ) : Extended;
   BEGIN
     Sinh:= 0.5*( EXP(XVal) - EXP(-XVal) );
   END;  { Function Sinh }


{ ------------------------------------------------------------------------------
|
|                              FUNCTION ARCSINH
|
|  This function evaluates the inverse hyperbolic sine function.
|
|  Algorithm     : Calculate the answer
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
|  Inputs        :
|    XVal        - Angle Value                                   any real
|
|  OutPuts       :
|    ArcSinh     - Result                                        any real
|
|  Locals        :
|    None.
|
|  Coupling      :
|    None.
|
 ------------------------------------------------------------------------------ }

FUNCTION ARCSinh              ( XVal                  : Extended ) : Extended;
   BEGIN

     ArcSinh:= LN( XVal + SQRT( XVal*XVal + 1.0 ) );

   END;  { Function ArcSinh }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                          FUNCTION TANH
|
|  This function evaluates the hyperbolic tangent function.
|
|  Algorithm      : Calculate the answer
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
|  Inputs         :
|    XVal         - Angle Value                               any real
|
|  OutPuts        :
|    Tanh         - Result                                    -1.0 to 1.0
|
|  Locals         :
|    None.
|
|  Coupling       :
|    None.
|
 ------------------------------------------------------------------------ }

FUNCTION Tanh                  ( XVal                    : Extended ) : Extended;
   BEGIN

     Tanh:= ( EXP(XVal) - EXP(-XVal) ) / ( EXP(XVal) + EXP(-XVal) );

   END;  { Function Tanh }


{ ------------------------------------------------------------------------
|
|                          FUNCTION ARCTANH
|
|  This function evaluates the inverse hyperbolic tangent function.
|
|  Algorithm      : Check for divide by zero and write error
|                     Calculate the answer if possible
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
|  Inputs         :
|    XVal         - Angle Value                               -1.0 to 1.0
|
|  OutPuts        :
|    ArcTanh      - Result                                    any real
|
|  Locals         :
|    Temp         - Temporary Extended Value
|
|  Constants      :
|    Small        - Tolerance
|    Undefined    - Flag value for an undefined quantity
|
|  Coupling       :
|    None.
|
 ------------------------------------------------------------------------ }

FUNCTION ARCTanh               ( XVal                    : Extended ) : Extended;
   Const
       Small     : Extended = 0.000001
       Undefined: Extended = 999999.1;
   VAR
       Temp : Extended;
   BEGIN
     IF 1.0 - ABS(XVal) < Small THEN
       BEGIN
         Temp:= Undefined;
         WriteLn( 'Error in ArcTanh Function ' );
       END
       ELSE
         Temp:= 0.5 * LN( (1.0 + XVal) / (1.0 - XVal) );

     ArcTanh:= Temp;
   END;  { Function ArcTanh }
{
```

```
}
{ ----------------------------------------------------------------------------
|
|                              FUNCTION DOT
|
|  This Function finds the dot product of tv › vectors.
|
|  Algorithm      : Calculate the answer directly
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
|  Inputs         :
|    Vec1         - Vector number 1
|    Vec2         - Vector number 2
|
|  OutPuts        :
|    Dot          - Result
|
|  Locals         :
|    None.
|
|  Coupling       :
|    None.
|
 ---------------------------------------------------------------------------- }

FUNCTION DOT                ( Vec1,Vec2                 : Vector ) : Extended;
   BEGIN
     DOT:= Vec1[1]*Vec2[1] + Vec1[2]*Vec2[2] + Vec1[3]*Vec2[3];
   END;  { Function Dot }



{ ----------------------------------------------------------------------------
|
|                              PROCEDURE CROSS
|
|  This procedure crosses two vectors.
|
|  Algorithm      : Calculate each vector component
|                   Find the magnitude of the answer
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
|  Inputs         :
|    Vec1         - Vector number 1
|    Vec2         - Vector number 2
|
|  OutPuts        :
|    OutVec       - Vector result of A x B
|
|  Locals         :
|    None.
|
|  Coupling       :
|    MAG            Magnitude of a vector
|
 ---------------------------------------------------------------------------- }

PROCEDURE CROSS             ( Vec1,Vec2                 : Vector;
                              VAR OutVec                : Vector );
   BEGIN
     OutVec[1]:= Vec1[2]*Vec2[3]-Vec1[3]*Vec2[2];
     OutVec[2]:= Vec1[3]*Vec2[1]-Vec1[1]*Vec2[3];
     OutVec[3]:= Vec1[1]*Vec2[2]-Vec1[2]*Vec2[1];

     MAG( OutVec );
   END;   { Procedure Cross }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                          PROCEDURE MAG
|
| This procedure finds the magnitude of a vector.  The tolerance is set to
|   0.000001, thus the 1.0E-12 for the squared test of underflows.
|
| Algorithm     : Find the squared sum of the terms
|                 Check to be sure there is no SQRT of 0.0
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
|
| Inputs        :
|   Vec       - Vector
|
| OutPuts       :
|   Vec       - Answer stored in fourth component
|
| Locals        :
|   None.
|
| Coupling      :
|   None.
|
 --------------------------------------------------------------------------- }

PROCEDURE MAG                ( VAR Vec                : Vector );
   VAR Temp: Extended;
   BEGIN
     Temp:= Vec[1]*Vec[1] + Vec[2]*Vec[2] + Vec[3]*Vec[3];

     IF ABS( Temp ) >= 1.0E-12 THEN
         Vec[4]:= SQRT( Temp )
       ELSE
         Vec[4]:= 0.0;
   END;  { Procedure Mag }

{ ---------------------------------------------------------------------------
|
|                          PROCEDURE NORM
|
| This Procedure calculates a unit vector given the original vector.  If a
|   zero vector is input, the vector is set to zero.
|
| Algorithm     : Find the magnitude of the input vector if not done
|                 Check if the magnitude is greater than zero
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   21 Aug 1988
|
| Inputs        :
|   Vec         - Vector
|
| OutPuts       :
|   OutVec      - Unit Vector
|
| Locals        :
|   i           - Index
|
| Constants     :
|   Small       - Tolerance factor
|
| Coupling      :
|   MAG           Magnitude of a vector
|
 --------------------------------------------------------------------------- }

PROCEDURE NORM               ( Vec                    : Vector;
                               VAR OutVec             : Vector );
   CONST
       Small : Extended = 0.000001;
   VAR
       i     : INTEGER;
   BEGIN
     MAG( Vec );
     IF Vec[4] > Small THEN
         FOR i:= 1 to 4 DO
             OutVec[i]:= Vec[i]/Vec[4]
       ELSE
         FOR i:= 1 to 4 DO
             OutVec[i]:= 0.0;

   END;  { Procedure Norm }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                              PROCEDURE ROT1
|
| This procedure performs a rotation about the 1st axis.
|
| Algorithm     : Store 3rd component for later use
|                 Calculate Sine and Cosine values to make more efficient
|                 Find the new vector
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs        :
|   Vec         - Input vector
|   XVal        - Angle of rotation                           rad
|
| OutPuts       :
|   OutVec      - Vector result
|
| Locals        :
|   c           - Cosine of the angle XVal
|   s           - Sine of the angle XVal
|   Temp        - Temporary Extended value
|
| Coupling      :
|   None.
|
 --------------------------------------------------------------------------- }

PROCEDURE ROT1                 ( Vec                       : Vector;
                                 XVal                      : Extended;
                                 VAR OutVec                : Vector );
   VAR
       c, s, Temp : Extended;
   BEGIN
     Temp:= Vec[3];
     c:= cos( XVal );
     s:= sin( XVal );

     OutVec[3]:= c*Vec[3] - s*Vec[2];
     OutVec[2]:= c*Vec[2] + s*Temp;
     OutVec[1]:= Vec[1];
     OutVec[4]:= Vec[4];
   END;  { Procedure Rot1 }

{
```

```
}
{ ---------------------------------------------------------------------------
|
|                              PROCEDURE ROT2
|
| This procedure performs a rotation about the 2nd axis.
|
| Algorithm     : Store 3rd component for later use
|                 Calculate Sine and Cosine values to make more efficient
|                 Find the new vector
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs        :
|   Vec         - Input vector
|   XVal        - Angle of rotation                              rad
|
| OutPuts       :
|   OutVec      - Vector result
|
| Locals        :
|   c           - Cosine of the angle XVal
|   s           - Sine of the angle XVal
|   Temp        - Temporary Extended value
|
| Coupling      :
|   None.
|
 --------------------------------------------------------------------------- }
PROCEDURE ROT2               ( Vec                       : Vector;
                               XVal                      : Extended;
                               VAR OutVec                : Vector );
   VAR
     c, s, Temp : Extended;
   BEGIN
     Temp:= Vec[3];
     c:= cos( XVal );
     s:= sin( XVal );

     OutVec[3]:= c*Vec[3] + s*Vec[1];
     OutVec[1]:= c*Vec[1] - s*Temp;
     OutVec[2]:= Vec[2];
     OutVec[4]:= Vec[4];
   END;  { Procedure Rot2 }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                          PROCEDURE ROT3
|
| This procedure performs a rotation about the 3rd axis.
|
| Algorithm     : Store 2nd component for later use
|                 Calculate Sine and Cosine values to make more efficient
|                 Find the new vector
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs        :
|   Vec         - Input vector
|   XVal        - Angle of rotation                              rad
|
| OutPuts       :
|   OutVec      - Vector result
|
| Locals        :
|   c           - Cosine of the angle XVal
|   s           - Sine of the angle XVal
|   Temp        - Temporary Extended value
|
| Coupling      :
|   None.
|
 --------------------------------------------------------------------------- }

PROCEDURE ROT3                ( Vec                        : Vector;
                                XVal                       : Extended;
                                VAR OutVec                 : Vector );
   VAR
     c, s, Temp : Extended;
   BEGIN
     Temp:= Vec[2];
     c:= cos( XVal );
     s:= sin( XVal );

     OutVec[2]:= c*Vec[2] - s*Vec[1];
     OutVec[1]:= c*Vec[1] + s*Temp;
     OutVec[3]:= Vec[3];
     OutVec[4]:= Vec[4];
   END;  { Procedure Rot3 }

{
```

```
}
{ -------------------------------------------------------------------------
|
|                             PROCEDURE ADDVEC
|
|  This procedure adds two vectors.
|
|  Algorithm      : Loop to find each component
|                   Find the magnitude of the vector
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
|  Inputs         :
|    Vec1         - Vector number 1
|    Vec2         - Vector number 2
|
|  OutPuts        :
|    OutVec       - Vector result of A + B
|
|  Locals         :
|    i            - Index
|
|  Coupling       :
|    MAG            Magnitude of a vector
|
 ------------------------------------------------------------------------- }

PROCEDURE ADDVEC              ( Vec1,Vec2                 : Vector;
                                VAR OutVec                : Vector );
   VAR
      i : Integer;
   BEGIN
     FOR i:=1 to 3 DO
         OutVec[i]:= Vec1[i] + Vec2[i];
     MAG( OutVec );
   END;  { Procedure AddVec }

{ -------------------------------------------------------------------------
|
|                             PROCEDURE ADD3VEC
|
|  This procedure adds three vectors.
|
|  Algorithm      : Loop to find each component
|                   Find the magnitude of the vector
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
|  Inputs         :
|    Vec1         - Vector number 1
|    Vec2         - Vector number 2
|    Vec3         - Vector number 3
|
|  OutPuts        :
|    OutVec       - Vector result of Vec1 + Vec2 + Vec3
|
|  Locals         :
|    i            - Index
|
|  Coupling       :
|    MAG            Magnitude of a vector
|
 ------------------------------------------------------------------------- }

PROCEDURE ADD3VEC             ( Vec1,Vec2,Vec3            : Vector;
                                VAR OutVec                : Vector );
   VAR
      i : Integer;
   BEGIN
     FOR i:=1 to 3 DO
         OutVec[i]:= Vec1[i] + Vec2[i] + Vec3[i];
     MAG( OutVec );
   END;  { Procedure Add3Vec }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                          PROCEDURE LNCOM1
|
| This procedure calculates the linear combination of a vector
|   multiplied by a constants.
|
| Algorithm      : Loop to find each combination
|                  Find the magnitude of the vector
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs         :
|   A1           - constant number
|   Vec          - Vector number
|
| OutPuts        :
|   OutVec       - Vector result of A1*Vec1 + A2*Vec2
|
| Locals         :
|   i            - Index
|
| Coupling       :
|   MAG            Magnitude of a vector
|
 --------------------------------------------------------------------------- }

PROCEDURE LNCOM1               ( A                            : Extended;
                                 Vec                          : Vector;
                                 VAR OutVec                   : Vector );
   VAR
       i : Integer;
   BEGIN
     FOR i:= 1 to 3 DO
         OutVec[i]:= A*Vec[i];
     MAG( OutVec );
   END;  { Procedure LnCom1 }


{ ---------------------------------------------------------------------------
|
|                          PROCEDURE LNCOM2
|
| This procedure calculates the linear combination of two vectors
|   multiplied by two different constants.
|
| Algorithm      : Loop to find each combination
|                  Find the magnitude of the vector
|
| Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs         :
|   A1           - constant number 1
|   A2           - constant number 2
|   Vec1         - Vector number 1
|   Vec2         - Vector number 2
|
| OutPuts        :
|   OutVec       - Vector result of A1*Vec1 + A2*Vec2
|
| Locals         :
|   i            - Index
|
| Coupling       :
|   MAG            Magnitude of a vector
|
 --------------------------------------------------------------------------- }

PROCEDURE LNCOM2               ( A1,A2                        : Extended;
                                 Vec1,Vec2                    : Vector;
                                 VAR OutVec                   : Vector );
   VAR
       i : Integer;
   BEGIN
     FOR i:= 1 to 3 DO
         OutVec[i]:= a1*Vec1[i] + a2*Vec2[i];
     MAG( OutVec );
   END;  { Procedure LnCom2 }

{
```

```
}
{ ------------------------------------------------------------------------------
|
|                          PROCEDURE LNCOM3
|
| This procedure calculates the linear combination of three vectors
|   multiplied by three different constants.
|
| Algorithm     : Loop to find each combination
|                 Find the magnitude of the vector
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs        :
|   A1          - constant number 1
|   A2          - constant number 2
|   A3          - constant number 3
|   Vec1        - Vector number 1
|   Vec2        - Vector number 2
|   Vec3        - Vector number 3
|
| OutPuts       :
|   OutVec      - Vector result of A1*Vec1 + A2*Vec2 + A3*Vec3
|
| Locals        :
|   i           - Index
|
| Coupling      :
|   MAG           Magnitude of a vector
|
 ------------------------------------------------------------------------------ }

PROCEDURE LNCOM3                ( A1,A2,A3                   : Extended;
                                  Vec1,Vec2,Vec3             : Vector;
                                  VAR OutVec                 : Vector );
   VAR
       i : Integer;
   BEGIN
     FOR i:= 1 to 3 DO
         OutVec[i]:= a1*Vec1[i] + a2*Vec2[i] + a3*Vec3[i];
     MAG( OutVec );
   END;  { Procedure LnCom3 }

{
```

```
}
{ ---------------------------------------------------------------------------
|
|                          PROCEDURE ANGLE
|
| This procedure calculates the angle between two vectors.  The output is
|   set to 999999.1 to indicate an undefined value.  Be SURE to check for
|   this at the output phase.
|
| Algorithm     : Check the denominator for a divide by zero
|                 Check for exactly 1.0 or -1.0 to avoid ArcCosine problems
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   14 Sep 1990
|
| Inputs        :
|   Vec1        - Vector number 1
|   Vec2        - Vector number 2
|
| OutPuts       :
|   Theta        - Angle between the two vectors             -Pi to Pi
|
| Locals        :
|   Temp        - Temporary REAL variable
|
| Constants     :
|   Undefined   - Undefined flag for a variable
|   Small       - Tolerance factor
|
| Coupling      :
|   DOT           Dot Product of two vectors
|   ArcCos        Arc Cosine function
|
 --------------------------------------------------------------------------- }

PROCEDURE ANGLE              ( Vec1,Vec2                 : Vector;
                               VAR Theta                 : Extended );
   CONST
     Small    : Extended = 0.000001;
     Undefined: Extended = 999999.1;
   VAR
       Temp : Extended;
   BEGIN
     IF Vec1[4]*Vec2[4] > SQR(Small) THEN
       BEGIN
         Temp:= DOT(Vec1,Vec2) / (Vec1[4]*Vec2[4]);
         IF ABS( Temp ) > 1.0 THEN
             Temp:= SGN(Temp) * 1.0;
         Theta:= ARCCOS( Temp );
       END
       ELSE
         Theta:= Undefined;
   END;  { Procedure Angle   }

{
```

```
}
{ ---------------------------------------------------------------------------
|
|                        PROCEDURE QUADRATIC
|
|  This procedure solves for the two roots of a quadratic equation.  There are
|    no restrictions on the coefficients, and imaginary results are passed
|    out as separate values.  The general form is y = ax2 + bx + c.
|
|  Algorithm     : Initialize all values
|                  Find discriminate
|                  Use discriminate value to separate the root calculations
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  18 Jun 1990
|
|  Inputs        :
|    a           - Coefficient of x squared term
|    b           - Coefficient of x term
|    c           - Constant
|
|  OutPuts       :
|    R1r         - Real portion of Root 1
|    R1i         - Imaginary portion of Root 1
|    R2r         - Real portion of Root 2
|    R2i         - Imaginary portion of Root 2
|
|  Locals        :
|    Discrim     - Discriminate b2 - 4ac
|
|  Constants     :
|    None.
|
|  Coupling      :
|    None.
|
|  References    :
|    Escobal       pg. 433-434
|
 -------------------------------------------------------------------------- }

PROCEDURE Quadratic           ( a,b,c                    : Extended;
                                VAR R1r,R1i,R2r,R2i      : Extended  );
   VAR
     Discrim : Extended;
   BEGIN
     { ----------------------  Initialize  ---------------------- }
     R1r:= 0.0;
     R1i:= 0.0;
     R2r:= 0.0;
     R2i:= 0.0;

     Discrim:= b*b - 4.0*a*c;

     { ----------------------  Real roots  ------------------------ }
     IF Discrim > 0.0 THEN
       BEGIN
         R1r:= ( -b + SQRT(Discrim) ) / ( 2.0*a );
         R2r:= ( -b - SQRT(Discrim) ) / ( 2.C*a );
       END
       ELSE
     { -------------------- Complex roots ------------------------ }
       BEGIN
         R1r:= -b / ( 2.0*a );
         R2r:= R1r;
         R1i:=  SQRT(-Discrim) / ( 2.0*a );
         R2i:= -SQRT(-Discrim) / ( 2.0*a );
       END;
   END;  { Procedure Quadratic }
{
```

```
}
{ ---------------------------------------------------------------------
|
|                         PROCEDURE CUBIC
|
| This procedure solves for the three roots of a cubic equation.  There are
|   no restrictions on the coefficients, and imaginary results are passed
|   out as separate values.  The general form is y = ax3 + bx2 + cx + d.  Note
|   that R1i will ALWAYS be ZERO since there is ALWAYS at least one REAL root.
|
| Algorithm     : Initialize variables
|                 Find correct coeffcients for the form of solution
|                 IF Delta is positive
|
|                     IF Delta is zero
|
|                        else
|                          find answers where Delta is negative
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  18 Jun 1990
|
| Inputs        :
|   a           - Coefficient of x cubed term
|   b           - Coefficient of x squared term
|   c           - Coefficient of x term
|   d           - Constant
|
| OutPuts       :
|   R1r         - Real portion of Root 1
|   R1i         - Imaginary portion of Root 1
|   R2r         - Real portion of Root 2
|   R2i         - Imaginary portion of Root 2
|   R3r         - Real portion of Root 3
|   R3i         - Imaginary portion of Root 3
|
| Locals        :
|   Temp1       - Temporary value
|   Temp2       - Temporary value
|   Root1       - Temporary value of the root
|   Root2       - Temporary value of the root
|   Root3       - Temporary value of the root
|   P           - Coefficient of x squared term where x cubed term is 1.0
|   Q           - Coefficient of x term where x cubed term is 1.0
|   R           - Coefficient of constant term where x cubed term is 1.0
|   Delta       - Discriminator for use with Cardans formula
|   E0          - Angle holder for trigonometric solution
|   Phi         - Angle used in trigonometric solution
|   CosPhi      - Cosine of Phi
|   SinPhi      - Sine of Phi
|
| Constants     :
|   Rad         - Radians per degree
|   Small       - Tolerance factor
|   OneThird    - 1.0/3.0
|
| Coupling      :
|   ATAN2         Arctangent including check for 180-360 deg
|
| References    :
|   Escobal       pg. 430-433
|
 -------------------------------------------------------------------- }
{
```

```
}
PROCEDURE Cubic                 ( a,b,c,d                        : Extended;
                                  VAR R1r,R1i,R2r,R2i,R3r,R3i: Extended  );
   CONST
     Rad       : Extended = 57.29577951308230;
     OneThird  : Extended = 1.0/3.0;
     Small     : Extended = 0.000001;
   VAR
      temp1, temp2, Root1, Root2, Root3, P, Q, R, Delta,
      E0, CosPhi, SinPhi, Phi : Extended;
   BEGIN
     { ----------------------    Initialize    ------------------------ }
     R1r  := 0.0;
     R1i  := 0.0;
     R2r  := 0.0;
     R2i  := 0.0;
     R3r  := 0.0;
     R3i  := 0.0;
     Root1:= 0.0;
     Root2:= 0.0;
     Root3:= 0.0;

     { ------------ Force coefficients into std form --------------- }
     P:= B/A;
     Q:= C/A;
     R:= D/A;

     a:= OneThird*( 3.0*Q - P*P );
     b:= (1.0/27.0)*( 2.0*P*P*P - 9.0*P*Q + 27.0*R );

     Delta:= (a*a*a/27.0) + (b*b/4.0);

     { ------------------- Use Cardans formula --------------------- }
     IF Delta > Small THEN
       BEGIN
         Temp1:= (-b*0.5)+SQRT(Delta);
         Temp2:= (-b*0.5)-SQRT(Delta);
         IF ABS(Temp1) > Small THEN
             Temp1:= SGN(Temp1)*POWER( SGN(Temp1)*Temp1,OneThird );
         IF ABS(Temp2) > Small THEN
             Temp2:= SGN(Temp2)*POWER( SGN(Temp2)*Temp2,OneThird );
         Root1:= Temp1 + Temp2;
         Root2:= -0.5*(Temp1 + Temp2);
         Root3:= -0.5*(Temp1 + Temp2);
         R2i:= -0.5*SQRT( 3.0 )*(Temp1 - Temp2);
         R3i:= -R2i;
       END
       ELSE
     { ---------------- Evaluate zero point ------------------------ }
       BEGIN
         IF ABS( Delta ) < Small  THEN
           BEGIN
             IF ABS(b) > Small THEN
               BEGIN
                 Root1:= -SGN(b)*2.0*POWER( SGN(b)*b/2.0,OneThird );
                 Root2:=  SGN(b)*POWER( SGN(b)*b/2.0,OneThird );
                 Root3:= Root2;
               END;
             { else let them be 0.0 since b is 0.0 }
           END
           ELSE
     { ---------------- Use trigonometric identities --------------- }
           BEGIN
             E0        := 2.0*SQRT(-a*OneThird);
             CosPhi:= (-b/(2.0*SQRT(-a*a*a/27.0)) );
             SinPhi:= SQRT( 1.0-CosPhi*CosPhi );
             Phi         := ATAN2( SinPhi,CosPhi );
             Root1:= E0*Cos( Phi*OneThird );
             Root2:= E0*Cos( Phi*OneThird + 120.0/Rad );
             Root3:= E0*Cos( Phi*OneThird + 240.0/Rad );
           END;
       END;

     R1r:= Root1 - P*OneThird;
     R2r:= Root2 - P*OneThird;
     R3r:= Root3 - P*OneThird;
   END;  { Procedure Cubic }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                           PROCEDURE QUARTIC
|
| This procedure solves for the four roots of a quartic equation.  There are
|   no restrictions on the coefficients, and imaginary results are passed
|   out as separate values.  The general form is y = ax4 + bx3 + cx2 + dx + e.
|
| Algorithm     :
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  18 Jun 1990
|
| Inputs        :
|   a           - Coeficient of x fourth term
|   b           - Coefficient of x cubed term
|   c           - Coefficient of x squared term
|   d           - Coefficient of x term
|   e           - Constant
|
| OutPuts       :
|   R1r         - Real portion of Root 1
|   R1i         - Imaginary portion of Root 1
|   R2r         - Real portion of Root 2
|   R2i         - Imaginary portion of Root 2
|   R3r         - Real portion of Root 3
|   R3i         - Imaginary portion of Root 3
|   R4r         - Real portion of Root 4
|   R4i         - Imaginary portion of Root 4
|
| Locals        :
|   Temp1       - Temporary value
|   Temp2       - Temporary value
|   Root1       - Temporary value of the root
|   Root2       - Temporary value of the root
|   Root3       - Temporary value of the root
|   s           - alternate variable
|   h           - Temporary value
|   hSqr        - h squared
|   hCube       - h Cubed
|   P           - Term in auxillary equation
|   Q           - Term in auxillary equation
|   R           - Term in auxillary equation
|   Delta       - Discriminator for use with Cardans formula
|   E0          - Angle holder for trigonometric solution
|   Phi         - Angle used in trigonometric solution
|   CosPhi      - Cosine of Phi
|   SinPhi      - Sine of Phi
|   RPrime      - Values of roots before final work
|   Temp        - Temporary variable in finding MAX RPrime
|   Eta         - Constant coefficient in quadratic solutions
|   Beta        - Constant coefficient in quadratic solutions
|
| Constants     :
|   Rad         - Radians per degree
|   Small       - Tolerance factor
|   OneThird    - 1.0/3.0
|
| Coupling      :
|   ATAN2         Arctangent including check for 180-360 deg
|
| References    :
|   Escobal       pg. 430-433
|
 -------------------------------------------------------------------------- }
{
```

```
PROCEDURE Quartic              ( a,b,c,d,e                    : Extended;
                                  VAR R1r,R1i,R2r,R2i,R3r,R3i,
                                      R4r,R4i                  : Extended  );
   CONST
     Rad      : Extended = 57.29577951308230;
     OneThird : Extended = 1.0/3.0;
     Small    : Extended = 0.000001;
   VAR
     Templ, Temp2, Root1, Root2, Root3, s, h, P, Q, R, Delta, E0,
     CosPhi, SinPhi, Phi, RPrime, hSqr, HCube, Eta, Beta, temp : Extended;
   BEGIN
     { ----------------------   Initialize   ---------------------- }
     R1r  := 0.0;
     R1i  := 0.0;
     R2r  := 0.0;
     R2i  := 0.0;
     R3r  := 0.0;
     R3i  := 0.0;
     R4r  := 0.0;
     R4i  := 0.0;
     Root1:= 0.0;
     Root2:= 0.0;
     Root3:= 0.0;
     { ------------ Force coefficients into std form --------------- }
     b:= B/A;
     c:= C/A;
     d:= D/A;
     e:= E/A;

     H:= -b/4;
     HSqr:= SQR( H );
     HCube:= HSqr * H;

     P:=                      6.0*HSqr   + 3.0*b*h + c;
     Q:=           4.0*HCube + 3.0*b*HSqr + 2.0*c*h + d;
     R:= h*HCube +  b*HCube +    c*HSqr   +     d*h  + e;

     a:=  (1.0/ 3.0)*( -P*P-12.0*R );
     b:=  (1.0/27.0)*( -2.0*P*P*P+72.0*P*R-27.0*Q*Q );
     s:= -(2.0/ 3.0)*P;

     Delta:= (a*a*a/27.0) + (b*b/4.0);

     IF ABS(Q) > Small THEN
       BEGIN
     { ------------------- Use Cardans formula --------------------- }
       IF Delta > Small THEN
         BEGIN
           Templ:= (-b*0.5)+SQRT(Delta);
           Temp2:= (-b*0.5)-SQRT(Delta);
           IF ABS(Templ) > Small THEN
               Templ:= SGN(Templ)*POWER( SGN(Templ)*Templ,OneThird );
           IF ABS(Temp2) > Small THEN
               Temp2:= SGN(Temp2)*POWER( SGN(Temp2)*Temp2,OneThird );
           Root1:= Templ + Temp2;
           Root2:= -0.5*(Templ + Temp2);
           Root3:= -0.5*(Templ + Temp2);
           R2i:= -0.5*SQRT( 3.0 )*(Templ - Temp2);
           R3i:= -R2i;
         END
         ELSE
     { ---------------- Evaluate zero point ----------------------- }
         BEGIN
           IF ABS( Delta ) < Small  THEN
             BEGIN
               IF ABS(b) > Small THEN
                 BEGIN
                   Root1:= -SGN(b)*2.0*POWER( SGN(b)*b/2.0,OneThird );
                   Root2:=  SGN(b)*POWER( SGN(b)*b/2.0,OneThird );
                   Root3:= Root2;
                 END;
               { else let them be 0.0 since b is 0.0 }
             END
             ELSE
     { ---------------- Use trigonometric identities --------------- }
             BEGIN
               E0         := 2.0*SQRT(-a*OneThird);
               CosPhi:= (-b/(2.0*SQRT(-a*a*a/27.0)) );
               SinPhi:= SQRT( 1.0-CosPhi*CosPhi );
               Phi        := ATAN2( SinPhi,CosPhi );
               Root1:= E0*Cos( Phi*OneThird );
               Root2:= E0*Cos( Phi*OneThird + 120.0/Rad );
               Root3:= E0*Cos( Phi*OneThird + 240.0/Rad );
             END;
         END;
{
```

```
}
    { ---------------- Find largest value of root ---------------- }
        RPrime:= Root1+s;
        IF (RPrime < Root2+s) and (ABS(R2i)<0.0001) THEN
            RPrime:= Root2+s;
        IF (RPrime < Root3+s) and (ABS(R3i)<0.0001) THEN
            RPrime:= Root3+s;

    { ------ Evaluate coefficients of two resulting Quadratics ---- }
        IF RPrime > Small THEN
          BEGIN
            Eta := 0.5*( P + RPrime - Q/SQRT(RPrime) );
            Beta:= 0.5*( P + RPrime + Q/SQRT(RPrime) );
          END
          ELSE
          BEGIN
            Eta := 0.5*P;
            Beta:= 0.5*P;
          END;

        Quadratic( 1.0, SQRT(RPrime),Eta,    R1r,R1i,R2r,R2i );
        Quadratic( 1.0,-SQRT(RPrime),Beta,   R3r,R3i,R4r,R4i );

      END { If Q > Small }
      ELSE
      BEGIN
    { ------- Case where solution reduces to a quadratic ---------- }
        Quadratic( 1.0,P,R,   R1r,R1i,R2r,R2i );
        R1r:= SQRT( R1r );
        R1i:= SQRT( R1i );
        R2r:= SQRT( R2r );
        R2i:= SQRT( R2i );
        R3r:= -R1r;
        R3i:= -R1i;
        R4r:= -R2r;
        R4i:= -R2i;
      END;

    R1r:= R1r + h;
    R2r:= R2r + h;
    R3r:= R3r + h;
    R4r:= R4r + h;
  END;  { Procedure Quartic }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                              PROCEDURE INITMATRIX
|
| This procedure initializes the matricies in pascal.  Notice the use of a
|   record structure.  This allows for arrays to be as large as needed,
|   provided memory exists.  Also note each time this is called, NEW is
|   invoked.  Thus, you can have Heap and memory problems if you don't use
|   DelMatrix to clear the pointer value!!
|
| Algorithm     : Loop through the Rows
|                   Loop through the cols
|                     Assign a NEW pointer and all record fields
|                     Build the doubly linked list of pointers
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   24 Jan 1990
|
| Inputs        :
|   Rows        - Number of rows for the matrix
|   Cols        - Number of columns for the matrix
|
| OutPuts       :
|   A           - Matrix to be initialized
|
| Locals        :
|   i           - Index
|   j           - index
|   NextData    - PTR to next data value
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
 --------------------------------------------------------------------------- }

PROCEDURE InitMatrix          ( Rows,Cols                 : Integer;
                                VAR A                     : Matrix );
   VAR
     i,j      : Integer;
     NextData : MatrixDataPtr;
   BEGIN
     NEW( A );
     A^.NumRows:= Rows;
     A^.NumCols:= Cols;
     A^.DPtr   := NIL;
     A^.Head   := NIL;
     A^.Tail   := NIL;

     FOR i:=1 to Rows DO
       BEGIN
         FOR j:= 1 to Cols DO
           BEGIN
             New( NextData );
             With NextData^ Do
               BEGIN
                 Index := (i-1)*A^.NumCols + j;
                 Number:= 0.0;
                 Next  := NIL;
                 Last  := NIL;
               END;
             IF A^.DPtr = NIL THEN
               BEGIN
                 NextData^.Last:= NIL;
                 A^.Head       := NextData;
               END
               ELSE
               BEGIN
                 NextData^.Last:= A^.Tail;
                 A^.Tail^.Next := NextData;
               END;
             A^.Tail      := NextData;
             A^.Tail^.Next:= NIL;
             A^.DPtr      := NextData;
           END;
       END;

   END;  { Procedure InitMatrix }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE DELMATRIX
|
|  This procedure deletes a matrix in pascal.  Notice the use of a
|    record structure.  It's important to clear the values when no longer
|    needed, or when they will be created again.
|
|  Algorithm      : Start at the head of the pointer list
|                   Loop while not pointing to NIL
|                       Dispose of each pointer
|                   Dispose of the last pointer
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Oct 1989
|
|  Inputs         :
|    A            - Matrix to be deleted
|
|  OutPuts        :
|    A            - Former Matrix
|
|  Locals         :
|    i            - Index
|    j            - index
|    Temp         - PTR to data value
|    NextTemp     - PTR to next data value
|
|  Constants      :
|    None.
|
|  Coupling       :
|    None.
|
 ------------------------------------------------------------------------ }

PROCEDURE DelMatrix           ( VAR A                     : Matrix );
   VAR
     i,j           : Integer;
     Temp,NextTemp : MatrixDataPtr;
   BEGIN
     Temp := A^.Head;

     WHILE Temp^.Next <> NIL DO
       BEGIN
         NextTemp:= Temp^.Next;
         DISPOSE( Temp );
         Temp:= NextTemp;
       END;

     DISPOSE( Temp );
     DISPOSE( A );
   END;  { Procedure DelMatrix }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                           FUNCTION GETVAL
|
| This function gets a value from the record structure.  The function is
|   necessary to decode the data.  It's set up to resemble the standard
|   array format, however, ['s are replaced by ('s.
|
| Algorithm     : Find the index where you desire to get data
|                 Loop forward or backward to the desired index
|                 Assign the value
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   24 Jan 1990
|
| Inputs        :
|   A           - Matrix
|   Rows        - Row number of desired element
|   Cols        - Col number of desired element
|
| OutPuts       :
|   GetXVal     - Value of the Row,Col point
|
| Locals        :
|   i           - Index
|   j           - index
|
| Constants     :
|   None.
|
| Coupling      :
|   None.
|
 ----------------------------------------------------------------------- }

FUNCTION GetVal              ( VAR A                    : Matrix;
                                Row,Col                 : Integer ) : Extended;
   VAR
       i,j : Integer;
   BEGIN
     j:= (Row-1)*A^.NumCols + Col;

     WHILE ( j > A^.DPtr^.Index ) and ( A^.DPtr^.Next <> NIL ) DO
         A^.DPtr:= A^.DPtr^.Next;

     WHILE ( j < A^.DPtr^.Index ) and ( A^.DPtr^.Last <> NIL ) DO
         A^.DPtr:= A^.DPtr^.Last;

     GetVal:= A^.DPtr^.Number;

   END;  { Function GetVal}
{
```

```
}
{ ----------------------------------------------------------------------------
|
|                              PROCEDURE ASSIGNVAL
|
|  This procedure assigns a value to the record structure.  This is
|    necessary to decode the data.  It's set up to resemble the standard
|    array format, however, ['s are replaced by ('s.
|
|  Algorithm     : Call getval to get the pointer at the correct index location
|                  Assign the value to the pointer variable record field
|
|  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   24 Jan 1990
|
|  Inputs        :
|    A           - Matrix
|    Rows        - Row number of desired element
|    Cols        - Col number of desired element
|    Number      - Value to assign at the desired location
|
|  OutPuts       :
|    A           - Matrix
|
|  Locals        :
|    i           - Index
|    j           - index
|
|  Constants     :
|    None.
|
|  Coupling      :
|    None.
|
 ---------------------------------------------------------------------------- }
PROCEDURE AssignVal           ( VAR A                    : Matrix;
                                Row,Col                  : Integer;
                                Number                   : Extended );
   VAR
      Temp : Extended;
   BEGIN
     Temp:= GetVal( A,Row,Col );
     A^.DPtr^.Number:= Number;
   END;  { Procedure AssignVal }
{
```

```
}
{ ----------------------------------------------------------------------------
|
|                         PROCEDURE MatMult
|
| This procedure multiplies two matricies together.
|
| Algorithm     : Initialize the pointers for the result matrix
|                 Loop through the Rows
|                    Loop through the Cols
|                       Loop through an index
|                          Multiply and add up each cell value
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs        :
|   Mat1        - Matrix number 1
|   Mat2        - Matrix number 2
|   Mat1r       - Matrix number 1 rows
|   Mat1c       - Matrix number 1 columns
|   Mat2c       - Matrix number 2 columns
|
| OutPuts       :
|   Mat3        - Matrix result of Mat1 * Mat2 of size mat1r x mat2c
|
| Locals        :
|   Row         - Row Index
|   Col         - Column Index
|   ktr         - Index
|
| Coupling      :
|   None.
|
| References    :
|   None.
|
 ---------------------------------------------------------------------------- }

PROCEDURE MatMult              ( Mat1,Mat2             : Matrix;
                                 Mat1r,Mat1c,Mat2c     : Integer ;
                                 VAR Mat3              : Matrix );
   VAR
      Row,Col,ktr : Integer;
   BEGIN
     InitMatrix( Mat1r,Mat2c,Mat3 );

     FOR Row:=1 to mat1r DO
         FOR Col:= 1 to mat2c DO
           BEGIN
             FOR ktr:= 1 to mat1c DO
                 AssignVal( Mat3,Row,Col, GetVal(Mat3,Row,Col)+GetVal(Mat1,Row,ktr)*
                                      GetVal(Mat2,ktr,Col) );
           END;
   END;  { Procedure MatMult }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE MatAdd
|
| This procedure adds two matricies together.
|
| Algorithm     : Initialize the pointers for the result matrix
|                 Loop through the Rows
|                    Loop through the Cols
|                        Add up each cell value
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   26 Jun 1989
|
| Inputs        :
|   Mat1        - Matrix number 1
|   Mat2        - Matrix number 2
|   Mat1r       - Matrix number 1 rows
|   Mat1c       - Matrix number 1 columns
|
| OutPuts       :
|   Mat3        - Matrix result of Mat1 + Mat2 of size mat1r x mat1c
|
| Locals        :
|   Row         - Row Index
|   Col         - Column Index
|
| Coupling      :
|   None.
|
| References    :
|   None.
|
 ------------------------------------------------------------------------ }

PROCEDURE MatAdd              ( Mat1,Mat2                : Matrix;
                                Mat1r,Mat1c              : Integer ;
                                VAR Mat3                 : Matrix     );
   VAR
       Row,Col  : Integer;
   BEGIN
     InitMatrix( Mat1r,Mat1c,Mat3 );

     FOR Row := 1 to Mat1r DO

         FOR Col :=1 to Mat1c DO

             AssignVal( Mat3,Row,Col, GetVal(Mat1,Row,Col) + GetVal(Mat2,Row,Col) );

   END;  { Procedure MatAdd }
{
```

```
}
{ -------------------------------------------------------------------
|
|                         PROCEDURE MATTRANS
|
| This procedure finds the transpose of a matrix.
|
| Algorithm     : Initialize the pointers for the result matrix
|                 Loop through the Rows
|                    Loop through the Cols
|                       Transpose each cell value
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs        :
|   Mat1        - Matrix number 1
|   Mat1r       - Matrix number 1 rows
|   Mat1c       - Matrix number 1 columns
|
| OutPuts       :
|   Mat2        - Matrix result of transpose Mat2
|
| Locals        :
|   Row         - Row Index
|   Col         - Column Index
|
| Coupling      :
|   None.
|
| References    :
|   None.
|
 ------------------------------------------------------------------- }

PROCEDURE MatTrans              ( Mat1                    : Matrix;
                                  Mat1r,Mat1c             : Integer ;
                                  VAR Mat2                : Matrix );
   VAR
      Row,Col : Integer;
   BEGIN
     InitMatrix( Mat1c,Mat1r,Mat2 );
     FOR Row :=1 to Mat1r DO
         FOR Col := 1 to Mat1c DO
             AssignVal( Mat2,Col,Row, GetVal(Mat1,Row,Col) );
   END;  { Procedure MatTrans }
{
```

```
}
{ -----------------------------------------------------------------------------
|
|                          PROCEDURE LUDECOMP
|
|  This procedure decomposes a matrix into an LU form.
|
|  Algorithm     :
|
|  Author        : Maj Tom Riggs      USAFA/DFAS  719-472-4109   27 Apr 1989
|                  Capt Dave Vallado USAFA/DFAS  719-472-4109    1 Aug 1989
|
|  Inputs        :
|    Order       - Order of matrix
|
|  OutPuts       :
|    LU          - LU decomposition matrix
|    Index       - Index vector for pivoting
|
|  Locals        :
|    i           - Index
|    j           - Index
|    k           - Index
|    Imax        - Pivot row pointer
|    Scale       - Scale factor vector
|    Sum         - Temporary Variables
|    AMax        - Temporary Variables
|    Dum         - Temporary Variables
|
|  Coupling      :
|    None.
|
|  References    :
|    Numerical Recipes - Flannery
|
 ---------------------------------------------------------------------------- }
{
```

```
}
PROCEDURE LUDeComp            ( VAR LU                   : Matrix;
                                VAR Index                : Intarray;
                                Order                    : INTEGER  );

   Const
     Small : Extended = 0.000001;
   Var
     I, J, K, IMax  : Integer;
     Scale          : Matrix;
     Sum, AMax, Dum : Extended;
   BEGIN
    InitMatrix( Order,1,Scale );
    IMax := 0;
    FOR I := 1 to Order do
      BEGIN
        AMax := 0.0;
        FOR J := 1 to Order do
            IF (Abs(GetVal( LU,i,j)) > AMax) THEN
                AMax := Abs(GetVal( LU,i,j));
        IF AMax = 0.0 then
          BEGIN
            Write(' Singular Matrix ');
            halt;
          END;
        AssignVal( Scale,i,1, 1.0 / AMax );
      END;
    FOR j := 1 to Order do
      BEGIN
        FOR i := 1 to j - 1 do
          BEGIN
            Sum := GetVal( LU,i,j);
            FOR k := 1 to i - 1 do
                Sum := Sum - GetVal( LU,i,k)*GetVal( LU,k,j);
            AssignVal( LU,i,j, Sum );
          END;
        AMax := 0.0;
        FOR i := j to Order do
          BEGIN
            Sum := GetVal( LU,i,j);
            FOR k := 1 to j - 1 do
                Sum := Sum - GetVal( LU,i,k)*GetVal( LU,k,j);
            AssignVal( LU,i,j,  Sum );
            Dum := GetVal( Scale,i,1 )*Abs(Sum);
            IF (Dum >= AMax) then
              BEGIN
                IMax := i;
                AMax := Dum;
              END;
          END;
        IF (j <> imax) then
          BEGIN
            FOR k := 1 to Order do
              BEGIN
                Dum := GetVal( LU,imax,K);
                AssignVal( LU,imax,k, GetVal( LU,j,k ) );
                AssignVal( LU,j,k,   Dum );
              END;
            AssignVal( Scale,imax,1, GetVal( Scale,j,1) );
          END;
        Index[j] := Imax;
        IF Abs(GetVal( LU,j,j)) < Small THEN
          BEGIN
            Write(' Matrix is Singular ');
            halt;
          END;
        IF (j <> Order) THEN
          BEGIN
            Dum := 1.0 / GetVal( LU,j,j);
            FOR i := j + 1 to Order do
                AssignVal( LU,i,j,  Dum*GetVal( LU,i,j) );
          END;
      END;
    DelMatrix( Scale );
  END; { Procedure LuDeCmp }
 {
```

```
}
{ ------------------------------------------------------------------------
|
|                          PROCEDURE LUBkSUB
|
| This procedure finds the inverse of a matrix using LU decomposition.
|
| Algorithm      :
|
| Author         : Maj Tom Riggs      USAFA/DFAS  719-472-4109   28 Apr 1989
|                  Capt Dave Vallado USAFA/DFAS  719-472-4109    1 Aug 1989
|
| Inputs         :
|   Order        - Order of matrix
|   LU           - LU decomposition matrix
|   Index        - Index vector for pivoting
|
| OutPuts        :
|   B            - Solution Vector
|
| Locals         :
|   i            - Index
|   j            - Index
|   I0           - Pointer to non-zero element
|   IPtr         - Pivot Rwo Pointer
|   Sum          - Temporary Variables
|
| Coupling       :
|   None.
|
| References     :
|   Numerical Recipes - Flannery
|
 ------------------------------------------------------------------------ }

PROCEDURE LUBkSub              ( LU                          : Matrix;
                                 Index                       : Intarray;
                                 Order                       : INTEGER;
                                 Var B                       : Matrix );
   VAR
      I, J, IPtr, I0: Integer;
      Sum        : Extended;
   BEGIN
     I0 := 0;
     FOR i := 1 to Order DO
       BEGIN
         IPtr := Index[i];
         Sum := GetVal( B,IPtr,1);
         AssignVal( B,Iptr,1, GetVal( B,i,1) );
         IF (I0 <> 0) then
             FOR j := I0 to i - 1 do
                 Sum := Sum - GetVal( LU,i,j)*GetVal( B,j,1)
           ELSE
             IF (Sum <> 0.0) THEN
                I0 := I;
         AssignVal( B,i,1,  Sum );
       END;

     AssignVal( B,Order,1, GetVal( B,Order,1)/GetVal( LU,Order,Order) );

     FOR i := (Order - 1) Downto 1 DO
       BEGIN
         Sum := GetVal( B,i,1);
         FOR j := i + 1 to Order do
            Sum := Sum - GetVal( LU,i,j)*GetVal( B,j,1);
         AssignVal( B,i,1, Sum / GetVal( LU,i,i) );
       END;
  END;
{
```

```
}
{ -------------------------------------------------------------------------
|
|                       PROCEDURE MATINVERSE
|
| This procedure finds the inverse of a matrix using LU decomposition.
|
| Algorithm     :
|
| Author        : Maj Tom Riggs     USAFA/DFAS  719-472-4109   28 Apr 1989
|                 Capt Dave Vallado USAFA/DFAS  719-472-4109    1 Aug 1989
|
| Inputs        :
|   Mat         - Matrix to invert
|   Order       - Order of matrix
|
| OutPuts       :
|   MatInv      - Inverted matrix
|
| Locals        :
|   i           - Index
|   j           - Index
|   Index       - Index vector for pivoting
|   LU          - LU decomposition matrix
|   B           - Operational vector to form MatInv
|
| Coupling      :
|   None.
|
| References    :
|   Numerical Recipes - Flannery
|
 ------------------------------------------------------------------------- }

PROCEDURE MatInverse          ( Mat                     : Matrix;
                                Order                   : INTEGER;
                                VAR MatInv              : Matrix );
   VAR
      I, J        : Integer;
      Index       : Intarray;
      LU,B        : Matrix;
   BEGIN
      InitMatrix( Order,Order, LU    );
      InitMatrix( Order,1,     B     );
      InitMatrix( Order,Order, MatInv );

      FOR i := 1 to Order DO
        BEGIN
          Index[i] := i;
          FOR j := 1 to Order DO
              AssignVal( LU,i,j, GetVal( Mat,i,j) );
        END;

      LUDeComp(LU, Index, Order);

      FOR j := 1 to Order DO
        BEGIN
          FOR i := 1 to Order DO
             IF (i = j) THEN
                 AssignVal( B,i,1, 1.0 )
               ELSE
                 AssignVal( B,i,1, 0.0 );

          LUBkSub(LU, Index, Order, B);

          FOR i := 1 to Order do
             AssignVal( MatInv,i,j,  GetVal( B,i,1) );
        END;
     DelMatrix( LU );
     DelMatrix( B );
   END; { Procedure MatInverse }
{
```

```
}
{ ---------------------------------------------------------------------------
|
|                          PROCEDURE PRINTMAT
|
|  This procedure prints a matrix.
|
|  Algorithm      : Write out the title for the matrix
|                   Loop through the rows and print out 1 row at a time
|
|  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109   11 Oct 1989
|
|  Inputs         :
|    Mat1         - Matrix to print out
|
|  OutPuts        :
|    None.
|
|  Locals         :
|    Row          - Row Index
|    Col          - Column Index
|
|  Coupling       :
|    None.
|
 --------------------------------------------------------------------------- }

PROCEDURE PrintMat             ( Mat1                        : Matrix;
                                 Title                       : STR64 );
   VAR
     Row,Col : Integer;
   BEGIN
     Writeln( Title );
     FOR Row:= 1 to Mat1^.NumRows DO
       BEGIN
         FOR Col:= 1 to Mat1^.NumCols DO
           BEGIN
             Write( ' ',GetVal( Mat1,Row,Col):12:8 );
             IF (Col MOD 6 = 0) and (Mat1^.NumCols > 6) THEN
               BEGIN
                 WriteLn;
                 Write( ' ' );
               END;
           END;
         WriteLn;
       END;
   END;  { Procedure PrintMat }
{
```

```
}
{ ------------------------------------------------------------------------
|
|                        FUNCTION DETERMINANT
|
| This function calculates the determinant value using L-U decompisition.
| The formula must have a NON-ZERO number in the 1,1 position.  IF the
| function senses a NON-ZERO number in row 1, it exchanges rowl for a row
| WITH a NON-ZERO number.
|
| Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
|
| Inputs        :
|   Order       - Order of determinaent (# of rows)
|   Mat1        - Matrix to find determinant of
|
| OutPuts       :
|   Determinant - Result
|
| Locals        :
|   i           - Index
|   j           - Index
|   k           - Index
|   n           - Index
|   Temp        -
|   D           -
|   Sum         -
|   L           -
|   U           -
|   Small       - Tolarar?s for comparing to 0.0
|
| Coupling      :
|   Marion        pg. 168-172, 126-127
|
------------------------------------------------------------------------- }
{
```

```
}
FUNCTION DETERMINANT          ( Matl                   : Matrix;
                                Order                  : Integer ) : Extended;
   CONST
     Small : Extended = 0.000001;
   VAR
       i,j,k,n        : Integer;
       Temp, D, Sum : Extended;
       L, U           : Matrix;
   BEGIN
     Sum:= 0.0;
   {-- Switch a non zero row to the first row --}
     IF ABS( GetVal( Matl,1,1 ) ) < Small THEN
       BEGIN
         j:= 1;
         WHILE j <= Order DO
           BEGIN
             IF ABS( GetVal( Matl,j,1 ) ) > Small THEN
               BEGIN
                 FOR k:= 1 to Order DO
                   BEGIN
                     Temp:=  GetVal( Matl,1,k );
                     AssignVal( Matl,1,k, GetVal( Matl,j,k ) );
                     AssignVal( Matl,j,k, Temp );
                   END;
                 j:= Order + 1;
               END;
             j:= j+1;
           END;
       END; { IF ABS(Matl[1,1]) < Small }

     FOR i:= 1 to Order DO
         AssignVal( L,i,1, GetVal( Matl,i,1 ) );
     FOR j:= 1 to Order DO
         AssignVal( U,1,j, GetVal( Matl,1,j ) / GetVal( L,1,1 ) );
     FOR j:= 2 to Order DO
       BEGIN
         FOR i:= j to Order DO
           BEGIN
             Sum:= 0.0;
             FOR k:= 1 to j-1 DO
                 Sum:= Sum+ GetVal( L,i,k )* GetVal( U,k,j );
             AssignVal( L,i,j, GetVal( Matl,i,j )- Sum );
           END;  { for i }
         AssignVal( U,j,j,  1.0 );
         IF j <> Order THEN
           BEGIN
             FOR i:= j+1 to Order DO
               BEGIN
                 Sum:= 0.0;
                 FOR k:= 1 to j-1 DO
                     Sum:= Sum +  GetVal( L,j,k)* GetVal( U,k,i );
                 AssignVal( U,j,i,( GetVal( Matl,j,i)-Sum)/ GetVal( L,j,j ) );
               END;  { for i }
           END;  { if j }
       END;  { for j }
     D:= 1.0;
     FOR i:= 1 to Order DO
         D:= D* GetVal( L,i,i);
     Determinant:= D;
   END;  { Function Determinant }

{
```

```
}

BEGIN
END.  { Unit Math }
```

APPENDIX C

FORTRAN SOURCE CODE

TECHNICAL ROUTINES

```
}
*

*     ----------------------------------------------------------------     *
*                                                                          *
*                         Module - ASTROLIB.FOR                            *
*                                                                          *
*  This file contains fundamental Astrodynamic Subroutines and Functions.  *
*                                                                          *
*          !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!             *
*      !!!!!!  USE IN ASTRO COURSES IS EXPRESSLY FORBIDDEN !!!!!!          *
*          !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!             *
*                                                                          *
*    *****************  NOTICE OF GOVERNMENT ORIGIN  ****************       *
*                                                                          *
*   This software has been developed by an employee of the United States   *
*   Government at the United States Air Force Academy, and is therefore    *
*   a work of the United States, and is NOT subject to copyright protection *
*   under the provisions of 17 U.S.C. 105.  ANY use of this work, or       *
*   inclusion in other works, must comply with the notice provisions of    *
*   17 U.S.C. 403.                                                         *
*                                                                          *
*    ****************************************************************      *
*                                                                          *
*  Author   : Capt Dave Vallado, USAFA Dept of Astronautics                *
*                                                                          *
*  Coml  719-472-4109, Autovon 259-4109 / 4110                             *
*                                                                          *
*                                                                          *
*  Current : 30 Jan 91 Capt Dave Vallado          VERSION 3.0              *
*                                                                          *
*  Changes : 28 Jan 91 Capt Dave Vallado                                   *
*                                                                          *
*            25 Jan 91 Capt Dave Vallado                                   *
*                        Update to Lahey Ver 3.0 / FORTRAN 90              *
*            20 Sep 90 Capt Dave Vallado          VERSION 2.1              *
*                        Misc fixes to speed up                            *
*            20 Apr 90 Capt Dave Vallado          VERSION 2.0              *
*                                                                          *
*            19 Dec 89 Capt Dave Vallado                                   *
*                        Version 1.4                                       *
*            24 Apr 89 Capt Dave Vallado                                   *
*                        Version 1.2                                       *
*            12 Feb 89 Capt Dave Vallado                                   *
*                        Standardized format                               *
*            28 Sep 88 Capt Dave Vallado                                   *
*                        Added HMS and DMS to Rad conversions              *
*            30 Aug 88 Capt Dave Vallado          VERSION 1.0              *
*                                                                          *
*     ----------------------------------------------------------------     *

*
* -------------------- Routines for time calculations ---------------
*
*
*    Subroutine JulianDay     ( Yr,Mon,D,H,M,S    JD                  )
*
*    Subroutine DayofYr2MDHMS ( Yr,Days,          Mon,D,H,M,S         )
*
*    Subroutine InvJulianDay  ( JD                Yr,Mon,D,H,M,S      )
*
*    Subroutine FindDays      ( Year,Month,Day,Hr,Min,Sec,  Days      )
*
*    Function   GSTime        ( JD                                    )
*
*    Function   GSTim0        ( Yr                                    )
*
*    Subroutine LSTime        ( Lon,JD            LST,GST             )
*
*    Subroutine SunRiseSet    ( JDate,Lat,Lon,WhichKind,
*                                                 UTSunRise,UTSunSet )
*
*    Subroutine HMStoUT       ( Hr,Min,Sec        UT                  )
*
*    Subroutine UTtoHMS       ( UT                Hr,Min,Sec          )
*
*    Subroutine HMStoRad      ( Hr,Min,Sec        HMS                 )
*
*    Subroutine RadtoHMS      ( HMS               Hr,Min,Sec          )
*
*    Subroutine DMStoRad      ( Deg,Min,Sec       DMS                 )
*
*    Subroutine RadtoDMS      ( DMS               Deg,Min,Sec         )
*
*
```

```
* ------------- Routines for Technical 2-Body calculations ------------
*
*     Subroutine Site          ( Lat,Alt,Lst,      RS,VS               )
*
*     Subroutine RVToPOS       ( Rho,Az,El,DRho,DAz,DEl
*                                                  Rhovec,DRhovec      )
*
*     Subroutine Track         ( Rho,Az,El,DRho,DAz,DEl,Lat,Lst,RS
*                                                  R,V                 )
*
*     Subroutine RAZEL         ( R,V,RS,Lat,Lst Rho,Az,El,
*                                                  DRho,DAz,DEl        )
*
*     Subroutine ELORB         ( R,V               P,A,E,Inc,Omega,
*                                                  Argp,Nu,M,U,L,CapPi)
*
*     Subroutine RandV         ( P,E,Inc,Omega,Argp,Nu,U,L,CapPi
*                                                  R,V                 )
*
*     Subroutine Gibbs         ( R1,R2,R3          V2,Theta,flt        )
*
*     Subroutine HerrGibbs     ( R1,R2,R3,JD1,JD2,JD3
*                                                  V2,Theta,flt)
*
*     Subroutine FindCandS     ( ZNew              CNew,SNew           )
*
*     Subroutine NewtonR       ( E,M               E0,Nu               )
*
*     Subroutine Kepler        ( Ro,Vo,Time        R,V                 )
*
*     Subroutine Gauss         ( R1,R2,DM,Time     V1,V2               )
*
*     Subroutine IJKtoLatLon   ( R,JD              GeoCnLat,Lon        )
*
*     Subroutine Sun           ( JD                RSun,RtAsc,Decl     )
*
*     Subroutine Moon          ( JD                RMoon,RtAsc,Decl    )
*
*     Subroutine PlanetRV      ( NumPlanet,JD,     R,V                 )
*
*     Function   Geocentric    ( Lat                                   )
*
*     Function   InvGeocentric ( Lat                                   )
*
*     Subroutine Sight         ( R1,R2,            LOS                 )
*
*     Subroutine Light         ( R,JD,             LIT                 )
*
*     Subroutine OMS2          ( Lat,Lon,Alt,Phi,Az,Speed,JD,   R,V    )
*
* ------------------ Routines for ICBM calculations -------------------
*
*     Subroutine RngAz         ( LLat,LLon,TLat,TLon,TOF
*                                                  Range, Az           )
*
*     Subroutine Path          ( LLat, LLon, Range, Az
*                                                  TLat, TLon          )
*
*     Subroutine Trajec        ( LLat,LLon,TLat,TLon,Rbo,Q,TypePhi
*                                                  Range,Phi,TOF,Az,
*                                                  ICPhi,ICVbo,ICRbo,VN )
*
```

```
}
* -------------- Routines for orbit transfer calculations -------------
*
*     Subroutine Hohmann       ( R1,R3,e1,e3,Nu1,Nu3,
*                                          DelVa,DelVb,TOF     )
*     Subroutine OneTangent    ( R1,R3,e1,e3,Nu1,Nu2,Nu3,
*                                          DelVa,DelVb,TOF     )
*     Subroutine GeneralCoplanar( R1,R3,e1,e2,e3,Nu1,Nu2a,Nu2b,Nu3,
*                                          DelVa,DelVb,TOF     )
*
*     Subroutine Rendezvous    ( Rcs1,Rcs2,PhaseI,NumRevs
*                                          PhaseF,WaitTime     )
*
*     Subroutine Interplanetary( R1,R2,Rbo,Rimpact,Mu1,Mut,Mu2,
*                                          Deltav1,Deltav2,Vbo,Vretro )
*
*     Subroutine Reentry       ( VRe,PhiRe,BC,H,  V,Decl,MaxDecl     )
*
*     Subroutine HillsR        ( R,V,alt,t,       R1,V1              )
*
*     Subroutine HillsV        ( R,alt,t,         V                  )
*

*
* ----------- Routines for Technical perturbed calculations -----------
*
* ---------------- and numerical integration techniques  --------------
*
*
*     Subroutine Target        ( RInt,VInt,RTgt,VTgt,Dm,TOF,
*                                          V1t,V2t,DV1,DV2     )
*
*     Subroutine PKepler       ( Ro,Vo,Time       R,V                )
*
*     Subroutine J2DragPert    ( Inc,E,N,NDot,    OmegaDOT,ArgpDOT,EDOT )
*
*     Subroutine Predict       ( JD,JDEpoch,no,Ndot,Eo,Edot,inco,Omegao,
*                                OmegaDot,Argpo,ArgpDot,Mo,Lat,Lon,Alt,
*                                          RtAsc,Decl,Rho,Az,El  )
*
*     Subroutine Deriv         ( Time,X,          XDot               )
*
*     Subroutine PertAccel     ( R,V,Time,WhichOne,BC,   APert       )
*
*     Subroutine PDeriv        ( Time,X,DerivType,BC, XDot           )
*
*     Subroutine RK4           ( ITime,DT,N,DerivType,BC, XDot       )
*
*     Subroutine ATMOS         ( R,Rho                               )
*
*     Subroutine CHEBY         ( ALT,             PAlt,RhoAlt        )
*
*

*  ---------------------------- CONSTANTS: -----------------------------------
*
*    Rad         = 57.29577951308230  Degrees per radian
*    HalfPi      =  1.57079632679490
*    Pi          =  3.14159265358979
*    TwoPi       =  6.28318530717959
*
*    OmegaEarth =  0.0588335906868878   Angular rotation of Earth (Rad/TU)
*    RadPerDay  =  6.30038809866574     Radians Earth rotates in 1 Sidereal day
*    TUMin      = 13.44685108204        Minutes in one Time Unit
*    TUDay      =  0.00933809102919444  Days per Time Unit
*    VKmPerSec  =  7.905366296149       KM/sec in one DU/TU
*
*    EESqrd     =  0.00669437999013     Eccentricity of Earth's shape squared
*    Flat       =  0.00335281066474 7352 Flatenning of the Earth
*
*    J2         =      0.00108263
*    J3         =     -0.00000254
*    J4         =     -0.00000161
*    GMS        = 332952.9364           Gravitational Parameter of Sun DU3/TU2
*    GMM        =      0.01229997       Gravitational Parameter of Moon DU3/TU2
*
*
```

```
* ---------------------------------------------------------------------------
*
*                        SUBROUTINE JULIANDAY
*
*  This subroutine finds the Julian date given the Year, Month, Day, and Time.
*    The Julian date is defined by each elapsed day since noon, 1 Jan 4713 BC.
*    Julian dates are measured from this epoch at noon so astronomers
*    observations may be performed on a single "day".  The year range is
*    limited since machine routines for 365 days a year and leap years are
*    valid in this range only.  This is due to the fact that leap years occur
*    only in years divisible by 4 and centuries whose number is evenly
*    divisible by 400. ( 1900 no, 2000 yes ... )
*
*  NOTE:  This Algorithm is taken from the 1988 Almanac for Computers,
*    Published by the U.S. Naval Observatory.  The algorithm is good for dates
*    between 1 Mar 1900 to 28 Feb 2100 since the last two terms (from the
*    Almanac) are commented out.
*
*  Algorithm     : Find the various terms of the expansion
*                  Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
*
*  Inputs        :
*    Yr          - Year                                  1900 .. 2100
*    Mon         - Month                                   1 .. 12
*    D           - Day                                     1 .. 28,29,30,31
*    H           - Universal Time Hour                     0 .. 23
*    M           - Universal Time Min                      0 .. 59
*    Sec         - Universal Time Sec                    0.0 .. 59.999
*
*  Outputs       :
*    JD          - Julian Date                           days from 4713 B.C.
*
*  Locals        :
*    Term1       - Temporary REAL value
*    Term2       - Temporary INTEGER value
*    Term3       - Temporary INTEGER value
*    UT          - Universal Time                           days
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    1988 Almanac for Computers  pg. B2
*    Escobal        pg. 17-19
*    Kaplan         pg. 329-330
*
* ---------------------------------------------------------------------------

      SUBROUTINE JulianDay ( Yr,Mon,D,H,M,S,  JD )
        IMPLICIT NONE
        INTEGER Yr,Mon,D,H,M
        REAL*8    S,JD

* -----------------------------  Locals  -----------------------------
        REAL*8 UT,Term1
        INTEGER Term2,Term3

* -------------------------  Implementation  -------------------------
        TERM1 = 367.0D0 * Yr
        TERM2 = INT( (7* (Yr+INT ( (Mon+9)/12) ) ) / 4 )
        TERM3 = INT( 275*Mon / 9 )
        UT = ( (S/60.0D0 + M ) / 60.0D0 + H ) / 24.0DC

        JD = (TERM1-TERM2+TERM3) + D + 1721013.5D0 + UT
      RETURN
      END
*
```

```
* ------------------------------------------------------------------------------
*
*                          SUBROUTINE DAYOFYR2MDHMS
*
*  This subroutine converts the day of the year, fractional days, to the month
*    day, hour, minute and second.
*
*  Algorithm     : Set up array for the number of days per month
*                  loop through a temp value while the value is < the days
*                  Perform integer conversions to the correct day and month
*                  Convert remainder into H M S using type conversions
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  26 Feb 1990
*
*  Inputs        :
*    Yr          - Year                                  1900 .. 2100
*    Days        - Julian Day of the year                 0.0 .. 366.9
*
*  Outputs       :
*    Mon         - Month                                 1 .. 12
*    D           - Day                                   1 .. 28,29,30,31
*    H           - Hour                                  0 .. 23
*    M           - Minute                                0 .. 59
*    Sec         - Second                                0.0 .. 59.999
*
*  Locals        :
*    dayyr       - Day of year                           days
*    Temp        - Temporary real values
*    IntTemp     - Temporary Integer value
*    i           - Index
*
*  Constants     :
*    LMonth(12)  - Integer Array containing the number of days per month
*
*  Coupling      :
*    None.
*
* ------------------------------------------------------------------------------

      SUBROUTINE DAYOFYR2MDHMS ( Yr,Days,  Mon,D,H,M,S )
        IMPLICIT NONE
        REAL*8 Days,S
        INTEGER Yr, Mon, D, H, M

* -----------------------------  Locals  ------------------------------
        INTEGER IntTemp,i,DayYr
        REAL*8 Temp, LMonth(12)

* ----------------- Set up array of days in month  --------------------
        DO i=1,12
            LMonth(i) = 31
        ENDDO
        LMonth( 2) = 28
        LMonth( 4) = 30
        LMonth( 6) = 30
        LMonth( 9) = 30
        LMonth(11) = 30
        DayYr = AINT( Days )

* -------------------- Find month and Day of month -------------------
        IF ( INT( MOD( Yr-1900,4 ) ).EQ.0 ) THEN
            LMonth(2)= 29
          ENDIF
        i= 1
        IntTemp= 0
        DO WHILE ( (DayYr.GT.IntTemp+LMonth(i)).and.(i.LT.12))
            IntTemp= IntTemp + LMonth(i)
            i= i+1
          ENDDO
        Mon = i
        D   = DayYr - IntTemp

* ------------------- Find hours minutes and seconds -----------------
        Temp= (Days - DayYr )*24.0D0
        H   = DINT( Temp )
        Temp= (Temp-H ) * 60.0D0
        M   = DINT( Temp )
        S   = (Temp-M ) *60.0D0

      RETURN
      END
*
```

```
* ----------------------------------------------------------------------------
*
*                              SUBROUTINE RAZEL
*
*  This Subroutine calculates Range Azimuth and Elevation and their rates given
*    the Geocentric Equatorial (IJK) Position and Velocity vectors.
*
*  Algorithm     : Find constant values
*                  Loop to find range and velocity vectors
*                  Rotate to find SEZ vectors
*                  Use if statments to find Az and El including special cases
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  27 Mar 1990
*
*  Inputs        :
*    R           - IJK Position Vector                     DU
*    V           - IJK Velocity Vector                     DU / TU
*    Lat         - Geodetic Latitude                       -Pi/2 to Pi/2 rad
*    LST         - Local Sidereal Time                     -2Pi to Pi rad
*    RS          - IJK Site Position Vector                DU
*
*  Outputs       :
*    Rho         - Satellite Range from site               DU
*    Az          - Azimuth                                 0 to 2Pi rad
*    El          - Elevation                               -Pi/2 to Pi/2 rad
*    DRho        - Range Rate                              DU / TU
*    DAz         - Azimuth Rate                            rad / TU
*    DEl         - Elevation rate                          rad / TU
*
*  Locals        :
*    RhoV        - IJK Range Vector from site              DU
*    DRhoV       - IJK Velocity Vector from site           DU / TU
*    RhoVec      - SEZ Range vector from site              DU
*    DRhoVec     - SEZ Velocity vector from site           DU
*    WCrossR     - Cross product result                    DU / TU
*    EarthRate   - IJK Earth's rotation rate vector        rad / TU
*    TempVec     - Temporary vector
*    Temp        - Temporary REAL value
*    Templ       - Temporary REAL value
*    Small       - Tolerance for roundoff errors
*    i           - Index
*
*  Constants     :
*    HalfPi      -                                         1.57079632679490
*    Pi          -                                         3.14159265358979
*    OmegaEarth  - Angular rotation of Earth (Rad/TU)      0.0588335906868878
*
*  Coupling      :
*    Mag           Magnitude of a vector
*    Cross         Cross product of two vectors
*    Rot3          Rotation about the 3rd axis
*    Rot2          Rotation about the 2nd axis
*    Dot           Dot product of two vectors
*
*  References    :
*    BMW           pg. 84-89, 100-101
*
* ----------------------------------------------------------------------------
*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE FINDDAYS
*
*  This subroutine finds the fractional days through a year given the year,
*    month, day, hour, minute and second.
*
*  Algorithm     : Set up array for the number of days per month
*                  Check for a leap year
*                  Loop to find the elapsed days in the year
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  11 Dec 1990
*
*  Inputs        :
*    Yr          - Year                                  1900 .. 2100
*    Mon         - Month                                    1 .. 12
*    D           - Day                                      1 .. 28,29,30,31
*    H           - Hour                                     0 .. 23
*    M           - Minute                                   0 .. 59
*    Sec         - Second                                 0.0 .. 59.999
*
*  OutPuts       :
*    days        - Day of year plus fraction of a day   days
*
*  Locals        :
*    i           - Index
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    None.
*
* ---------------------------------------------------------------------------

      SUBROUTINE FindDays( Year,Month,Day,Hr,Min,Sec,  Days )
        IMPLICIT NONE
        INTEGER Year,Month,Day,Hr,Min
        REAL*8  Sec, Days

* ------------------------------  Locals  ------------------------------
        INTEGER i,LMonth(12)

* ----------------- Set up array of days in month  -------------------
        DO i=1,12
            LMonth(i) = 31
        ENDDO
        LMonth( 2) = 28
        LMonth( 4) = 30
        LMonth( 6) = 30
        LMonth( 9) = 30
        LMonth(11) = 30

        IF ( INT( MOD( Year-1900,4 ) ).EQ.0 ) THEN
           LMonth(2)= 29
         ENDIF
       i   = 1
       Days= 0.0D0
       DO WHILE ((i.lt.Month).and.( i.lt.12 ))
           Days= Days + LMonth(i)
           i= i + 1
       ENDDO
       Days= Days + Day + Hr/24.0D0 + Min/1440.0D0 + Sec/86400.0D0

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                           FUNCTION GSTIME
*
*  This function finds the Greenwich Sidereal time.  Notice just the integer
*    part of the Julian Date is used for the Julian centuries calculation.
*
*  Algorithm     : Perform expansion calculation to obtain the answer
*                  Check the answer for the correct quadrant and size
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Feb 1989
*
*  Inputs        :
*    JD          - Julian Date                              days from 4713 B.C.
*
*  Outputs       :
*    GSTime      - Greenwich Sidereal Time                  0 to 2Pi rad
*
*  Locals        :
*    Temp        - Temporary variable for Reals             rad
*    Tu          - Julian Centuries from 1 Jan 2000
*
*  Constants     :
*    TwoPi       -                                          6.28318530717959
*    RadPerDay   - Rads Earth rotates in 1 Solar Day        6.30038809866574
*
*  Coupling      :
*    None.
*
*  References    :
*    1988 Astronomical Almanac  pg. B6
*    Escobal     pg. 18 - 21
*    Explanatory Supplement pg. 73-75
*    Kaplan      pg. 330-332
*    BMW         pg. 103-104
*
* -----------------------------------------------------------------------------

      REAL*8 FUNCTION GSTime ( JD )
        IMPLICIT NONE
        REAL*8 JD

* ---------------------------- Locals ------------------------------
        REAL*8 Temp, Tu, RadPerDay, TwoPi

* ------------------------ Implementation ------------------------
        RadPerDay= 6.30038809866574D0
        TwoPi    = 6.28318530717959D0

        Tu  = ( DINT(JD) + 0.5D0 - 2451545.0D0 ) / 36525.0D0
        Temp= 1.753368559D0 + 628.3319705D0*Tu+6.770708127D-06*Tu**2+
     &        RadPerDay*DBLE( JD-DINT(JD)-0.5 )

* ------------------------ Check quadrants ------------------------
        Temp  = DMOD( Temp,TwoPi )
        IF ( Temp.LT.0.0D0 ) THEN
            Temp = Temp + TwoPi
          ENDIF
        GSTime= Temp
      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                              FUNCTION GSTIM0
*
*  This function finds the Greenwich Sidereal time at the beginning of a year.
*    This formula is derived from the Astonomical Almanac and is good only for
*    0 hr UT, 1 Jan of a year.
*
*  Algorithm     : Find the Julian Date Ref 4713 BC
*                  Perform expansion calculation to obtain the answer
*                  Check the answer for the correct quadrant and size
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Nov 1989
*
*  Inputs        :
*    Yr          - Year                                    1988, 1989, etc.
*
*  Outputs       :
*    GSTim0      - Greenwich Sidereal Time                 0.0 to 2Pi rad
*
*  Locals        :
*    JD          - Julian Date                             days from 4713 B.C.
*    Temp        - Temporary variable for Reals            rad
*    Tu          - Julian Centuries from 1 Jan 2000
*
*  Constants     :
*    TwoPi                                                  6.28318530717959
*
*  Coupling      :
*    None.
*
*  References    :
*    1988 Astronomical Almanac  pg. B6
*    Escobal     pg. 18 - 21
*    Explanatory Supplement pg. 73-75
*    Kaplan      pg. 330-332
*    BMW         pg. 103-104
*
* -----------------------------------------------------------------------------

      REAL*8 FUNCTION GSTim0 ( Yr )
        IMPLICIT NONE
        INTEGER Yr

* ----------------------------  Locals  ------------------------------
        REAL*8 JD, Temp, Tu, TwoPi

* ------------------------  Implementation  --------------------------
        TwoPi    = 6.28318530717959D0

        JD  = 367.0D0 * Yr - ( INT((7*(Yr+INT(10/12)))/4) ) +
     &                       ( INT(275/9) ) + 1721014.5D0
        Tu  = ( INT(JD) + 0.5D0 - 2451545.0D0 ) / 36525.0D0
        Temp= 1.753368559D0 + 628.3319705D0*Tu + 6.770708127D-06*Tu**2

* ------------------------  Check quadrants  -------------------------
        Temp  = DMOD( Temp,TwoPi )
        IF ( Temp.LT.0.0D0 ) THEN
            Temp = Temp + TwoPi
          ENDIF
        GSTim0= Temp
      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE LSTIME
*
*  This subroutine finds the Local Sidereal time at a given location.
*
*  Algorithm     : Find GST through the routine
*                  Find LST and check for size and quadrant
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
*
*  Inputs        :
*    Lon         - Site longitude (WEST -)                -2Pi to 2Pi rad
*    JD          - Julian Date                            days from 4713 B.C.
*
*  Outputs       :
*    LST         - Local Sidereal Time                    0.0 to 2Pi rad
*    GST         - Greenwich Sidereal Time                0.0 to 2Pi rad
*
*  Locals        :
*    None.
*
*  Constants     :
*    TwoPi                                                 6.28318530717959
*
*  Coupling      :
*    GSTime      - Finds the Greenwich Sidereal Time
*
*  References    :
*    Escobal     pg. 18 - 21
*    Kaplan      pg. 330-332
*    BMW         pg. 99 -100
*
* ---------------------------------------------------------------------------

      SUBROUTINE LSTime ( Lon,JD,  LST,GST )
        IMPLICIT NONE
        REAL*8 Lon,JD, LST, GST
        EXTERNAL GSTime

* ---------------------------- Locals ------------------------------
        REAL*8 TwoPi,GSTime

* ------------------------ Implementation  ------------------------
        TwoPi  = 6.28318530717959D0

        GST = GSTime( JD )
        LST = Lon + GST

* ------------------------ Check quadrants  ------------------------
        LST = DMOD( LST,TwoPi )
        IF ( LST.LT.0.0D0 ) THEN
            LST = LST + TwoPi
          ENDIF
      RETURN
      END

*
```

```
* -------------------- -----------------------------------------------------
*
*                          SUBROUTINE SUNRISESET
*
*  This subroutine finds the Universal time for Sunrise and Sunset given the
*    day and site location.  Note the use of degrees and radians since the
*    Almanac presents the algorithm in these units.
*
*  Algorithm     : Use a case statement to set the angle from the sun to site
*                  Find days, and then the values for UT times
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  13 Jan 1991
*
*  Inputs        :
*    JDate       - Julian Date                            days from 4713 B.C.
*    Lat         - Site latitude (SOUTH -)                -Pi/2 to Pi/2 rad
*    Lon         - Site longitude (WEST -)                 -2Pi to 2Pi rad
*    WhichKind   - Character for which rise/set            'S' 'C' 'N' 'A'
*
*  OutPuts       :
*    UTSunRise   - Universal time of sunrise at lat-lon       hrs
*    UTSunSet    - Universal time of sunset at lat-lon        hrs
*
*  Locals        :
*    t           - Days from the beginning of the year
*
*  Constants     :
*    Rad           Radians per degree
*
*  Coupling      :
*    InvJulianDay  Finds the Yr Da Mn Hr Mi Se from the Julian Date
*    FindDays      Finds the days from 1 Jan of a year
*
*  References    :
*    Almanac For Computers 1990 pg. B5-B6
*
* ---------------------------------------------------------------------------
*
```

```
}
      SUBROUTINE SUNRISESET(JDate,Lat,Lon,WhichKind,UTSunRise,UTSunSet)
        IMPLICIT NONE
        REAL*8 JDate,Lat,Lon,UTSunRise,UTSunSet
        CHARACTER WhichKind
        REAL*8   Z,t,m,l,ra,sindelta,delta,h,sec,days,Rad,TwoPi,Pi
        INTEGER  year,month,day,hr,min

        Rad   = 57.29577951308230D0
        TwoPi = 6.28318530717959D0
        Pi    = 3.14159265358979D0
        IF (WhichKind.eq.'S' ) THEN
            Z= (90.0D0+50.0D0/60.0D0 )/Rad
        ELSEIF (WhichKind.eq.'C') THEN
            Z=  96.0D0 / Rad
        ELSEIF (WhichKind.eq.'N') THEN
            Z= 102.0D0 / Rad
        ELSEIF (WhichKind.eq.'A') THEN
            Z= 108.0D0 / Rad
        ENDIF

        CALL InvJulianDay( JDate,  Year,Month,Day,Hr,Min,Sec )
        CALL FindDays( Year,Month,Day,Hr,Min,Sec,  Days )

* ----------------------------- Sunrise -----------------------------
        t = Days + (6.0D0 - Lon*Rad/15.0D0)/24.0D0
        M = 0.985600D0*t - 3.289D0
        L = M + 1.916D0*DSin( M/Rad ) + 0.020D0*DSin( 2.0D0*M/Rad ) +
     &      282.634D0
        L = DMOD( L,360.0 )
        Ra= DATan( 0.91746D0*DTan(L/Rad) )
        IF (Ra.lt.0.0D0) THEN
            Ra= Ra + TwoPi
          ENDIF
        IF ( (L.gt.180.0D0).and.(Ra.lt.Pi) ) THEN
            Ra= Ra + Pi
          ENDIF
        IF ( (L.lt.180.0D0).and.(Ra.gt.Pi) ) THEN
            Ra= Ra - Pi
          ENDIF
        SinDelta= 0.39782D0*DSin( L/Rad )
        Delta   = DASin( SinDelta )
        H= DACos( (DCos(z) - SinDelta*DSin(Lat)) /
     &             (DCos(Delta)*DCos(Lat) ) )
        H= TwoPi - H
        t= H*Rad/15.0D0 + RA*Rad/15.0D0 - 0.065710D0*t - 6.622D0
        T= DMOD( T, 24.0D0 )

        UTSunRise= T - Lon*Rad/15.0D0
        UTSunRise= DMOD( UTSunRise, 24.0D0 )
        IF (UTSunRise.lt.0.0D0) THEN
            UTSunRise= 24.0D0 + UTSunRise
          ENDIF

* ----------------------------- Sunset ------------------------------
        t = Days + (18.0D0 - Lon*Rad/15.0D0)/24.0D0
        M = 0.985600D0*t - 3.289D0
        L = M + 1.916D0*DSin( M/Rad ) + 0.020D0*DSin( 2.0D0*M/Rad ) +
     &      282.634D0
        L = DMOD( L,360.0 )
        Ra= DATan( 0.91746D0*DTan(L/Rad) )
        IF (Ra.lt.0.0D0) THEN
            Ra= Ra + TwoPi
          ENDIF
        IF ( (L.gt.180.0D0).and.(Ra.lt.Pi) ) THEN
            Ra= Ra + Pi
          ENDIF
        IF ( (L.lt.180.0D0).and.(Ra.gt.Pi) ) THEN
            Ra= Ra - Pi
          ENDIF
        SinDelta= 0.39782D0*DSin( L/Rad )
        Delta   = DASin( SinDelta )
        H= DACos( (DCos(z) - SinDelta*DSin(Lat)) /
     &             (DCos(Delta)*DCos(Lat) ) )
        H= TwoPi - H
        t= H*Rad/15.0D0 + RA*Rad/15.0D0 - 0.065710D0*t - 6.622D0
        T= DMOD( T, 24.0D0 )

        UTSunSet= T - Lon*Rad/15.0D0
        UTSunSet= DMOD( UTSunSet, 24.0D0 )
        IF (UTSunSet.lt.0.0D0) THEN
            UTSunSet= 24.0D0 + UTSunSet
          ENDIF

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                              SUBROUTINE HMSTOUT
*
*  This subroutine converts Hours, Minutes and Seconds into Universal Time.
*
*  Algorithm     : Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
*
*  Inputs        :
*    Hr          - Hours                              0 .. 24    ex.    2
*    Min         - Minutes                            0 .. 59    ex.   39
*    Sec         - Seconds                          0.0 .. 59.99 ex.  57.29
*
*  Outputs       :
*    UT          - Universal Time                    HrMin.Sec    ex.239.5729
*
*  Locals        :
*    None.
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
* -----------------------------------------------------------------------------

      SUBROUTINE HMStoUT ( Hr,Min,Sec,  UT )
        IMPLICIT NONE
        REAL*8 UT,Sec
        INTEGER Hr,Min

* -------------------------  Implementation   ------------------------
        UT = Hr*100.0D0 + Min + Sec/100.0D0

      RETURN
      END

* -----------------------------------------------------------------------------
*
*                              SUBROUTINE UTTOHMS
*
*  This subroutine converts Universal Time into Hours, minutes and seconds.
*
*  Algorithm     : Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
*
*  Inputs        :
*    UT          - Universal Time                    HrMin.Sec    ex.239.5729
*
*  Outputs       :
*    Hr          - Hours                              0 .. 24    ex.    2
*    Min         - Minutes                            0 .. 59    ex.   39
*    Sec         - Seconds                          0.0 .. 59.99 ex.  57.29
*
*  Locals        :
*    None.
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
* -----------------------------------------------------------------------------

      SUBROUTINE UTtoHMS ( UT,  Hr,Min,Sec )
        IMPLICIT NONE
        REAL*8 UT,Sec
        INTEGER Hr,Min

* -------------------------  Implementation   ------------------------
        Hr = IDINT( UT/100.0D0 )
        Min= IDINT( UT-Hr*100.0D0 )
        Sec= ( UT-DINT(UT) ) * 100.0D0
      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                            SUBROUTINE HMSTORAD
*
*  This subroutine converts Hours, minutes and seconds into radians.  Notice
*    the conversion 0.2617 is simply the radian equivalent of 15 degrees.
*
*  Algorithm     : Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   8 Sep 1988
*
*  Inputs        :
*    Hr          - Hours                               0 .. 24    ex.   10
*    Min         - Minutes                             0 .. 59    ex.   15
*    Sec         - Seconds                           0.0 .. 59.99 ex.  30.00
*
*  Outputs       :
*    HMS         - Result                               rad       ex. 2.6856253
*
*  Locals        :
*    None.
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
* -----------------------------------------------------------------------------

      SUBROUTINE HMStoRad ( Hr,Min,Sec,  HMS )
        IMPLICIT NONE
        REAL*8 HMS,Sec
        INTEGER Hr,Min
* ------------------------  Implementation   ------------------------
        HMS= ( Hr + Min/60.0D0 + Sec/3600.0D0 )*0.261799387D0

      RETURN
      END

* -----------------------------------------------------------------------------
*
*                            SUBROUTINE RADTOHMS
*
*  This subroutine converts radians into Hours, minutes and seconds.  Notice
*    the conversion 0.2617 is simply the radian equivalent of 15 degrees.
*
*  Algorithm     : Convert incoming radians to hours
*                  Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   8 Sep 1988
*
*  Inputs        :
*    HMS         - Result                               rad       ex. 2.6856253
*
*  Outputs       :
*    Hr          - Hours                               0 .. 24    ex.   10
*    Min         - Minutes                             0 .. 59    ex.   15
*    Sec         - Seconds                           0.0 .. 59.99 ex.  30.00
*
*  Locals        :
*    Temp        - Temporary variable to hold and change HMS value
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
* -----------------------------------------------------------------------------

      SUBROUTINE RadtoHMS ( HMS,  Hr,Min,Sec )
        IMPLICIT NONE
        REAL*8 HMS,Sec
        INTEGER Hr,Min
        REAL*8 Temp
* ------------------------  Implementation   ------------------------
        Temp = HMS / 0.261799387D0
        Hr   = IDINT( Temp )
        Min  = IDINT( (Temp-Hr)*60.0D0 )
        Sec  = (Temp-Hr-Min/60.0D0 ) * 3600.0D0

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                          SUBROUTINE HMSTORAD
*
*  This subroutine converts Hours, minutes and seconds into radians.  Notice
*    the conversion 0.2617 is simply the radian equivalent of 15 degrees.
*
*  Algorithm     : Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   8 Sep 1988
*
*  Inputs        :
*    Hr          - Hours                               0 .. 24    ex.   10
*    Min         - Minutes                             0 .. 59    ex.   15
*    Sec         - Seconds                           0.0 .. 59.99 ex.  30.00
*
*  Outputs       :
*    HMS         - Result                               rad       ex. 2.6856253
*
*  Locals        :
*    None.
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
* ---------------------------------------------------------------------------

      SUBROUTINE HMStoRad ( Hr,Min,Sec,  HMS )
        IMPLICIT NONE
        REAL*8 HMS,Sec
        INTEGER Hr,Min
* -------------------------  Implementation   ------------------------
        HMS= ( Hr + Min/60.0D0 + Sec/3600.0D0 )*0.261799387D0

      RETURN
      END

* ---------------------------------------------------------------------------
*
*                          SUBROUTINE RADTOHMS
*
*  This subroutine converts radians into Hours, minutes and seconds.  Notice
*    the conversion 0.2617 is simply the radian equivalent of 15 degrees.
*
*  Algorithm     : Convert incoming radians to hours
*                  Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   8 Sep 1988
*
*  Inputs        :
*    HMS         - Result                               rad       ex. 2.6856253
*
*  Outputs       :
*    Hr          - Hours                               0 .. 24    ex.   10
*    Min         - Minutes                             0 .. 59    ex.   15
*    Sec         - Seconds                           0.0 .. 59.99 ex.  30.00
*
*  Locals        :
*    Temp        - Temporary variable to hold and change HMS value
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
* ---------------------------------------------------------------------------

      SUBROUTINE RadtoHMS ( HMS,  Hr,Min,Sec )
        IMPLICIT NONE
        REAL*8 HMS,Sec
        INTEGER Hr,Min
        REAL*8 Temp
* -------------------------  Implementation   ------------------------
        Temp = HMS / 0.261799387D0
        Hr   = IDINT( Temp )
        Min  = IDINT( (Temp-Hr)*60.0D0 )
        Sec  = (Temp-Hr-Min/60.0D0 ) * 3600.0D0

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                          SUBROUTINE SITE
*
*  This Subroutine finds the position and velocity vectors for a site.  The
*    answer is returned in the Geocentric Equatorial (IJK) coordinate system.
*
*  Algorithm     : Set up constants
*                  Find x and z values
*                  Find position vector directly
*                  Call cross to find the velocity vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs        :
*    Lat         - Geodetic Latitude                     -Pi/2 to Pi/2 rad
*    Alt         - Altitude                              DU
*    LST         - Local Sidereal Time                   -2Pi to 2Pi rad
*
*  Outputs       :
*    RS          - IJK Site position vector              DU
*    VS          - IJK Site velocity vector              DU / TU
*
*  Locals        :
*    EarthRate   - IJK Earth's rotation rate vector      rad / TU
*    SinLat      - Variable containing  sin( Lat )       rad
*    Temp        - Temporary REAL value
*    x           - x component of site vector            DU
*    z           - z component of site vector            DU
*
*  Constants     :
*    EESqrd      - Eccentricity of Earth's shape squared  0.00669437999013
*    OmegaEarth  - Angular rotation of Earth (Rad/TU)     0.0588335906868878
*
*  Coupling      :
*    Mag           Magnitude of a vector
*    Cross         Cross product of two vectors
*
*  References    :
*    Escobal       pg. 26 - 29  (includes Geocentric Lat formulation also)
*    Kaplan        pg. 334-336
*    BMW           pg. 94 - 98
*
* ------------------------------------------------------------------------

      SUBROUTINE Site ( Lat,Alt,Lst,  RS,VS )
        IMPLICIT NONE
        REAL*8 Lat, Alt, LST, RS(4), VS(4)

* ---------------------------- Locals -------------------------------
        REAL*8 SinLat, Temp, x, z, EarthRate(4),OmegaEarth,EESqrd

* ------------------------ Initialize Variables ----------------------
        OmegaEarth   = 0.0588335906868878D0
        EESqrd       = 0.00669437999013D0
        SinLat       = DSIN( Lat )
        EarthRate(1)= 0.0D0
        EarthRate(2)= 0.0D0
        EarthRate(3)= OmegaEarth

* -------------- Find x and z components of site vector  -------------
        Temp   = DSQRT( 1.0D0 - ( EESqrd*SinLat**2 ) )
        x      = ( ( 1.0D0/Temp ) + Alt )*DCOS( Lat )
        z      = ( ((1.0D0-EESqrd)/Temp) + Alt )*SinLat

* ---------------------- Find Site position vector  -----------------
        RS(1)  = x * DCOS( Lst )
        RS(2)  = x * DSIN( Lst )
        RS(3)  = z
        CALL MAG( RS )

* ------------------- Find Site velocity vector  --------------------
        CALL CROSS( EarthRate,RS,VS )
      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                              SUBROUTINE SITE
*
*  This Subroutine finds the position and velocity vectors for a site.  The
*    answer is returned in the Geocentric Equatorial (IJK) coordinate system.
*
*  Algorithm     : Set up constants
*                  Find x and z values
*                  Find position vector directly
*                  Call cross to find the velocity vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs        :
*    Lat         - Geodetic Latitude                       -Pi/2 to Pi/2 rad
*    Alt         - Altitude                                 DU
*    LST         - Local Sidereal Time                     -2Pi to 2Pi rad
*
*  Outputs       :
*    RS          - IJK Site position vector                 DU
*    VS          - IJK Site velocity vector                 DU / TU
*
*  Locals        :
*    EarthRate   - IJK Earth's rotation rate vector         rad / TU
*    SinLat      - Variable containing  sin( Lat )          rad
*    Temp        - Temporary REAL value
*    x           - x component of site vector               DU
*    z           - z component of site vector               DU
*
*  Constants     :
*    EESqrd      - Eccentricity of Earth's shape squared   0.00669437999013
*    OmegaEarth  - Angular rotation of Earth (Rad/TU)      0.0588335906868878
*
*  Coupling      :
*    Mag           Magnitude of a vector
*    Cross         Cross product of two vectors
*
*  References    :
*    Escobal       pg. 26 - 29  (includes Geocentric Lat formulation also)
*    Kaplan        pg. 334-336
*    BMW           pg. 94 - 98
*
* ------------------------------------------------------------------------

      SUBROUTINE Site ( Lat,Alt,Lst,  RS,VS )
        IMPLICIT NONE
        REAL*8 Lat, Alt, LST, RS(4), VS(4)

* ----------------------------  Locals  ------------------------------
        REAL*8 SinLat, Temp, x, z, EarthRate(4),OmegaEarth,EESqrd

* ----------------------- Initialize Variables -----------------------
        OmegaEarth  = 0.0588335906868878D0
        EESqrd      = 0.00669437999013D0
        SinLat      = DSIN( Lat )
        EarthRate(1)= 0.0D0
        EarthRate(2)= 0.0D0
        EarthRate(3)= OmegaEarth

* -------------  Find x and z components of site vector  ------------
        Temp   = DSQRT( 1.0D0 - ( EESqrd*SinLat**2 ) )
        x      = ( ( 1.0D0/Temp ) + Alt )*DCOS( Lat )
        z      = ( ((1.0D0-EESqrd)/Temp) + Alt )*SinLat

* ---------------------- Find Site position vector  -----------------
        RS(1)  = x * DCOS( Lst )
        RS(2)  = x * DSIN( Lst )
        RS(3)  = z
        CALL MAG( RS )

* ------------------ Find Site velocity vector  ---------------------
        CALL CROSS( EarthRate,RS,VS )
      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                              SUBROUTINE TRACK
*
*  This Subroutine finds range and velocity vectors in the Geocentric Equatorial
*    (IJK) system given the following input from a radar site.
*
*  Algorithm      : Find constant values
*                   Find SEZ vectors from RVToPOS
*                   Rotate to find IJK vectors
*
*  Author         : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs         :
*    Rho          - Satellite range from site             DU
*    Az           - Azimuth                               0.0 to 2Pi rad
*    El           - Elevation                            -Pi/2 to Pi/2 rad
*    DRho         - Range Rate                            DU / TU
*    DAz          - Azimuth Rate                          rad / TU
*    DEl          - Elevation rate                        rad / TU
*    Lat          - Geodetic Latitude                    -Pi/2 to Pi/2 rad
*    LST          - Local Sidereal Time                  -2Pi to 2Pi rad
*    RS           - IJK Site position vector              DU
*
*  Outputs        :
*    R            - IJK Satellite position vector         DU
*    V            - IJK Satellite velocity vector         DU / TU
*
*  Locals         :
*    WCrossR      - Cross product result                  DU / TU
*    RhoVec       - SEZ range vector from site            DU
*    DRhoVec      - SEZ velocity vector from site         DU / TU
*    TempVec      - Temporary vector
*    RhoV         - IJK range vector from site            DU
*    DRhoV        - IJK velocity vector from site         DU / TU
*    ERate        - IJK Earth's rotation rate vector      rad / TU
*
*  Constants        :
*    HalfPi       -                                       1.57079632679490
*    OmegaEarth   - Angular rotation of Earth (Rad/TU)    0.0588335906868878
*
*  Coupling       :
*    RVToPos        Find R and V from site in Topocentric Horizon (SEZ) system
*    Cross          Cross product of two vectors
*    Addvec         Add two vectors together
*    Rot3           Rotation about the 3rd axis
*    Rot2           Rotation about the 2nd axis
*    MAG            Magnitude of a vector
*
*  References     :
*    BMW            pg. 85-89, 100-101
*
* ------------------------------------------------------------------------

      SUBROUTINE Track ( Rho,Az,El,DRho,DAz,DEl,Lat,Lst,RS,  R,V )
        IMPLICIT NONE
        REAL*8 Rho,Az,El,DRho,DAz,DEl,Lat,Lst,RS(4),R(4),V(4)
* ----------------------------- Locals -----------------------------
        REAL*8 WCrossR(4), RhoVec(4), DRhoVec(4), TempVec(4), RhoV(4),
     &         DRhoV(4), ERate(4),HalfPi,OmegaEarth

* ------------------------ Initialize Variables -----------------------
        HalfPi     = 1.57079632679490D0
        OmegaEarth = 0.0588335906868878D0
        ERate(1)   = 0.0D0
        ERate(2)   = 0.0D0
        ERate(3)   = OmegaEarth
* ------------- Find SEZ range and velocity vectors -----------------
        CALL RVTOPOS( Rho,Az,El,DRho,DAz,DEl,RhoVec,DRhoVec )

* ---------------- Perform SEZ to IJK transformation ----------------
        CALL ROT2( RhoVec ,Lat-HalfPi, TempVec )
        CALL ROT3( TempVec,    -LST   , RhoV )
        CALL ROT2( DRhoVec,Lat-HalfPi, TempVec )
        CALL ROT3( TempVec,    -LST   , DRhoV )

* --------------- Find IJK range and velocity vectors ---------------
        CALL ADDVEC( RhoV,RS,R )
        CALL CROSS( ERate,R ,WCrossR )
        CALL ADDVEC( DRhoV,WCrossR,V )
      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                            SUBROUTINE RAZEL
*
*  This Subroutine calculates Range Azimuth and Elevation and their rates given
*    the Geocentric Equatorial (IJK) Position and Velocity vectors.
*
*  Algorithm     : Find constant values
*                  Loop to find range and velocity vectors
*                  Rotate to find SEZ vectors
*                  Use if statments to find Az and El including special cases
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  27 Mar 1990
*
*  Inputs        :
*    R           - IJK Position Vector                   DU
*    V           - IJK Velocity Vector                   DU / TU
*    Lat         - Geodetic Latitude                     -Pi/2 to Pi/2 rad
*    LST         - Local Sidereal Time                   -2Pi to Pi rad
*    RS          - IJK Site Position Vector              DU
*
*  Outputs       :
*    Rho         - Satellite Range from site             DU
*    Az          - Azimuth                               0 to 2Pi rad
*    El          - Elevation                             -Pi/2 to Pi/2 rad
*    DRho        - Range Rate                            DU / TU
*    DAz         - Azimuth Rate                          rad / TU
*    DEl         - Elevation rate                        rad / TU
*
*  Locals        :
*    RhoV        - IJK Range Vector from site            DU
*    DRhoV       - IJK Velocity Vector from site         DU / TU
*    RhoVec      - SEZ Range vector from site            DU
*    DRhoVec     - SEZ Velocity vector from site         DU
*    WCrossR     - Cross product result                  DU / TU
*    EarthRate   - IJK Earth's rotation rate vector      rad / TU
*    TempVec     - Temporary vector
*    Temp        - Temporary REAL value
*    Temp1       - Temporary REAL value
*    Small       - Tolerance for roundoff errors
*    i           - Index
*
*  Constants     :
*    HalfPi      -                                       1.57079632679490
*    Pi          -                                       3.14159265358979
*    OmegaEarth  - Angular rotation of Earth (Rad/TU)    0.0588335906868878
*
*  Coupling      :
*    Mag           Magnitude of a vector
*    Cross         Cross product of two vectors
*    Rot3          Rotation about the 3rd axis
*    Rot2          Rotation about the 2nd axis
*    Dot           Dot product of two vectors
*
*  References    :
*    BMW           pg. 84-89, 100-101
*
* ---------------------------------------------------------------------------
*
```

```
      SUBROUTINE RAZEL ( R,V,RS,Lat,Lst,  Rho,Az,El,DRho,DAz,DEl )
        IMPLICIT NONE
        REAL*8 R(4),V(4),RS(4),Lat,Lst,Rho,Az,El,DRho,DAz,DEl
        EXTERNAL DOT

* ------------------------------ Locals ------------------------------
        REAL*8   RhoV(4), DRhoV(4), RhoVec(4), DRhoVec(4), WCrossR(4),
     &           ERate(4), TempVec(4), Temp, Small,HalfPi,Pi,Templ,
     &           OmegaEarth,Dot
        INTEGER  i

* ------------------------ Initialize Variables ------------------------
        Pi          = 3.14159265358979D0
        HalfPi      = 1.57079632679490D0
        OmegaEarth  = 0.0588335906868878D0
        ERate(1)    = 0.0D0
        ERate(2)    = 0.0D0
        ERate(3)    = OmegaEarth
        Small       = 0.000001D0

* ----------- Find IJK range vector from site to satellite -----------
        CALL CROSS( ERate,R,WCrossR )
        DO i=1,3
            RhoV(i) =  R(i) - RS(i)
            DRhoV(i)=  V(i) - WCrossR(i)
        ENDDO
        CALL MAG( RhoV )
        Rho= RhoV(4)

* ----------------- Convert to SEZ for calculations -----------------
        CALL ROT3( RhoV ,     LST    ,  TempVec )
        CALL ROT2( TempVec,HalfPi-Lat,   RhoVec    )
        CALL ROT3( DRhoV,     LST    ,  TempVec )
        CALL ROT2( TempVec,HalfPi-Lat,  DRhoVec    )

* ----------------- Calculate Azimuth and Elevation -----------------
        Temp = DSQRT( RhoVec(1)**2 + RhoVec(2)**2 )
        IF ( DABS( RhoVec(2) ).LT.Small ) THEN
            IF ( Temp.LT.Small ) THEN
                Templ = DSQRT( DRhoVec(1)**2 + DRhoVec(2)**2 )
                Az = DATAN2( DRhoVec(2)/Templ , -DRhoVec(1)/Templ )
              ELSE
                IF ( RhoVec(1).GT.0.0D0 ) THEN
                    Az = Pi
                  ELSE
                    Az = 0.0D0
                  ENDIF
              ENDIF
          ELSE
            Az = DATAN2( RhoVec(2)/Temp , -RhoVec(1)/Temp )
          ENDIF
        IF ( Temp.LT.Small ) THEN
            El = HalfPi
          ELSE
            El = DATAN2( RhoVec(3)/Rho , Temp/Rho      )
          ENDIF

* ---------- Calculate Range, Azimuth and Elevation rates ------------
        DRho= DOT( RhoV,DRhoV ) / Rho
        IF ( DABS(Temp).GT.Small ) THEN
            DAz = ( DRhoVec(1)*RhoVec(2) - DRhoVec(2)*RhoVec(1) ) /
     &            (Temp**2)
          ELSE
            DAz= 0.0D0
          ENDIF
        IF ( DABS(Temp).GT.0.000000001D0 ) THEN
            DEl = ( DRhoVec(3) - DRho*DSIN( El ) ) / Temp
          ELSE
            DEl= 0.0D0
          ENDIF
      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                            SUBROUTINE ELORB
*
*  This Subroutine finds the classical orbital elements given the Geocentric
*    Equatorial Position and Velocity vectors.  Special cases for equatorial
*    and circular orbits are also handled.  IF elements are Infinite, they
*    are set to 999999.9.  If elements are Undefined, they are set to 999999.1.
*    Be sure to check for these during output!!
*
*  Algorithm     : Initialze variables
*                  If the Hbar magnitude exists, continue, otherwise exit and
*                       assign undefined values
*                    Find vectors and values
*                    Determine the type of orbit with IF statements
*                    Find angles depending on the orbit type
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs        :
*    R           - IJK Position vector                       DU
*    V           - IJK Velocity vector                       DU / TU
*
*  Outputs       :
*    P           - Semi-latus rectum                         DU
*    A           - semi-major axis                           DU
*    Ecc         - eccentricity
*    Inc         - inclination                               0.0 to Pi rad
*    Omaga       - Longitude of Ascending Node               0.0 to 2Pi rad
*    Argp        - Argument of Perigee                       0.0 to 2Pi rad
*    Nu          - True anomaly                              0.0 to 2Pi rad
*    M           - Mean Anomaly                              0.0 to 2Pi rad
*    U           - Argument of Latitude      (CI)            0.0 to 2Pi rad
*    L           - True Longitude            (CE)            0.0 to 2Pi rad
*    CapPi       - Longitude of Periapsis    (EE)            0.0 to 2Pi rad
*
*  Locals        :
*    Hbar        - Angular Momentum H Vector                 DU2 / TU
*    Ebar        - Eccentricity     E Vector
*    Nbar        - Line of Nodes    N Vector
*    c1          - V**2 - u/R
*    RDotV       - R Dot V
*    c3          - Hk unit vector
*    Small       - Tolerance for roundoff errors
*    SME         - Specfic Mechanical Energy                 DU2 / TU2
*    i           - index
*    E           - Eccentric Anomaly                         rad
*    D           - Parabolic Eccentric Anomaly               rad
*    F           - Hyperbolic Eccentric Anomaly              rad
*    Temp        - Temporary value
*    TypeOrbit   - Type of orbit                             EE, EI, CE, CI
*
*  Constants     :
*    HalfPi      -                                           1.57079632679490
*    Pi          -                                           3.14159265358979
*    TwoPi       -                                           6.28318530717959
*    Infinite    - Flag for an infintite element             999999.9
*    Undefined   - Flag for an undefined element             999999.1
*
*  Coupling      :
*    MAG           Magnitude of a vector
*    CROSS         Cross product of two vectors
*    DOT           DOT product of two vectors
*    DACOSH        Inverse Double Precision Hyperbolic Cosine Function
*    ANGLE         Angle between two vectors
*
*  References    :
*    BMW           pg. 58 - 71
*    Escobal       pg. 104-107
*    Kaplan        pg. 29 - 37
*
* ------------------------------------------------------------------------
*
```

```
      SUBROUTINE ELORB ( R,V,  P,A,Ecc,Inc,Omega,Argp,Nu,M,U,L,CapPi )
        IMPLICIT NONE
        REAL*8 R(4),V(4),P,A,Ecc,Inc,Omega,Argp,Nu,M,U,L,CapPi
        EXTERNAL DOT,DACOSH

* ---------------------------- Locals ----------------------------
        REAL*8 c1,RDotV,c3,Small,SME,Hbar(4),Ebar(4),Nbar(4),TwoPi,
     &         HalfPi,Pi,Dot,Undefined,Infinite,E,F,D,DACosh,Temp
        INTEGER i
        CHARACTER*2 TypeOrbit

* ----------------------- Initialize Variables ----------------------
        Pi        = 3.14159265358979D0
        HalfPi    = 1.57079632679490D0
        TwoPi     = 6.28318530717959D0
        Small     = 0.000001D0
        Infinite  = 999999.9D0
        Undefined = 999999.1D0
        CALL MAG( R )
        CALL MAG( V )

* --------------------- Find H N and E vectors   --------------------
        CALL CROSS( R,V,Hbar )

        IF ( HBar(4).GT.Small ) THEN
            Nbar(1)= -Hbar(2)
            Nbar(2)=  Hbar(1)
            Nbar(3)=   0.0D0
            CALL MAG( Nbar )
            c1= V(4)**2 - 1.0D0/R(4)
            RDotV= DOT( R,V )
            DO i= 1,3
                Ebar(i)= c1*R(i) - RDotV*V(i)
            ENDDO
            CALL MAG( Ebar )

* ------------------ Find a e and semi-latus rectum  ----------------
            SME= ( V(4)**2 / 2.0D0 ) - ( 1.0D0/R(4) )
            IF ( DABS(SME).GT.Small ) THEN
                A= -1.0D0 / (2.0D0*SME)
              ELSE
                A= Infinite
              ENDIF
            Ecc= Ebar(4)
            P= HBar(4)**2

* ----------------------- Find inclination  -------------------------
            c3= HBar(3)/HBar(4)
            IF ( DABS( DABS(c3)-1.0D0 ).LT.Small ) THEN
                IF ( DABS(Hbar(3)).GT.0.0D0 ) THEN
                    c3= DSIGN( 1.0D0,Hbar(3) )
                  ENDIF
              ENDIF
            Inc= DACOS( c3 )

* -------------- Determine type of orbit for later use  -------------
            TypeOrbit= 'EI'
            IF ( Ecc.LT.Small ) THEN
* ---------------------- Circular Equatorial ------------------------
                IF (( Inc.LT.Small ).or.( DABS(Inc-Pi).LT.Small )) THEN
                    TypeOrbit= 'CE'
                  ELSE
* ---------------------- Circular Inclined --------------------------
                    TypeOrbit= 'CI'
                  ENDIF
              ELSE
* ---------- Elliptical, Parabolic, Hyperbolic Equatorial -----------
                IF (( Inc.LT.Small ).or.( ABS(Inc-Pi).LT.Small )) THEN
                    TypeOrbit= 'EE'
                  ENDIF
              ENDIF
*
```

```
* --------------- Find Longitude of Ascending Node ------------------
           IF ( NBar(4).GT.Small ) THEN
               Temp= Nbar(1) / NBar(4)
               IF ( DABS(Temp).gt.1.0D0 ) THEN
                   Temp = DSIGN( 1.0D0,Temp )
                 ENDIF
               Omega= DACOS( Temp )
               IF ( NBar(2).LT.0.0D0 ) THEN
                   Omega= TwoPi - Omega
                 ENDIF
             ELSE
               Omega= Undefined
             ENDIF

* -------------------- Find Argument of perigee ----------------------
           IF ( TypeOrbit.eq.'EI' ) THEN
               CALL ANGLE( NBar,EBar, Argp )
               IF (EBar(3).LT.0.0D0) THEN
                   Argp= TwoPi - Argp
                 ENDIF
             ELSE
               Argp= Undefined
             ENDIF

* ----------------- Find True Anomaly at Epoch    --------------------
           IF ( TypeOrbit(1:1).eq.'E' ) THEN
               CALL ANGLE( EBar,R, Nu )
               IF (RDotV.LT.0.0D0) THEN
                   Nu= TwoPi - Nu
                 ENDIF
             ELSE
               Nu= Undefined
             ENDIF

* ---------- Find Argument of Latitude - Circular Inclined ----------
           IF ( TypeOrbit.EQ.'CI' ) THEN
               CALL ANGLE( NBar,R, U )
               IF (R(3).LT.0.0D0) THEN
                   U= TwoPi - U
                 ENDIF
             ELSE
               U= Undefined
             ENDIF

* ------- Find Longitude of Perigee - Elliptical Equatorial -----------
           IF (( EBar(4).GT.Small ).and.( TypeOrbit.EQ.'EE' )) THEN
               Temp = EBar(1)/EBar(4)
               IF ( DABS(Temp).gt.1.0D0 ) THEN
                   Temp = DSIGN( 1.0D0,Temp )
                 ENDIF
               CapPi= DACOS( Temp )
               IF (Ebar(2).LT.0.0D0) THEN
                   CapPi= TwoPi - CapPi
                 ENDIF
               IF (Inc.GT.HalfPi) THEN
                   CapPi= TwoPi - CapPi
                 ENDIF
             ELSE
               CapPi= Undefined
             ENDIF

* ----------- Find True Longitude - Circular Equatorial ---------------
           IF (( R(4).GT.Small ).and.( TypeOrbit.EQ.'CE' )) THEN
               Temp = R(1)/R(4)
               IF ( DABS(Temp).gt.1.0D0 ) THEN
                   Temp = DSIGN( 1.0D0,Temp )
                 ENDIF
               L= DACOS( Temp )
               IF (R(2).LT.0.0D0) THEN
                   L= TwoPi - L
                 ENDIF
               IF (Inc.GT.HalfPi) THEN
                   L= TwoPi - L
                 ENDIF
             ELSE
               L= Undefined
             ENDIF
*
```

```
* ----------------- Find Mean Anomaly for all orbits -----------------

* ----------------------------Hyperbolic ------------------------------
        IF ((Ecc-1.0D0).GT.Small) THEN
            F= DACOSH( (Ecc+DCos(Nu))/(1.0D0+Ecc*DCos(Nu)) )
            M= Ecc*DSinh( F ) - F
          ELSE

* --------------------------- Parabolic --------------------------------
            IF ( (DABS( Ecc-1.0D0 )).LT.Small ) THEN
                D = DSQRT( p ) * DTan( Nu )
                M = (1.0D0/6.0D0)*( 3.0D0*p*D + D**3 )
              ELSE

* --------------------------- Elliptical ------------------------------
                IF ( Ecc.GT.Small ) THEN
                    Temp= 1.0D0 + ecc*DCos(Nu)
                    IF ( DABS(Temp).lt.Small ) THEN
                        M = 0.0D0
                      ELSE
                        c1 = ( DSQRT(1.0D0-Ecc**2)*DSin(Nu) ) / Temp
                        c3 = ( Ecc + DCos(Nu) ) / Temp
                        IF ( DABS(c1).gt.1.0D0 ) THEN
                            c1 = DSIGN( 1.0D0,c1 )
                          ENDIF
                        IF ( DABS(c3).gt.1.0D0 ) THEN
                            c3 = DSIGN( 1.0D0,c3 )
                          ENDIF
                        E = DATan2( c1,c3 )
                        M = E - Ecc*DSin( E )
                      ENDIF
                  ELSE

* ---------------------------  Circular  -----------------------------
                      IF ( TypeOrbit.EQ.'CE' ) THEN
                          M = L
                        ELSE
                          M = U
                        ENDIF
                  ENDIF
              ENDIF
          ENDIF

        IF ( M.Lt.0.0D0 ) THEN
            M = M + TwoPi
          ENDIF
*
*       Write( *,20 ) 'H = ',Hbar(1),Hbar(2),Hbar(3),Hbar(4)
*       Write( *,20 ) 'N = ',Nbar(1),Nbar(2),Nbar(3),Nbar(4)
*       Write( *,20 ) 'E = ',Ebar(1),Ebar(2),Ebar(3),Ebar(4)
*       Write( *,* ) 'SME= ',SME,' DU2/TU2'
*   20  FORMAT( A4,2X,4(F13.7) )
*
         ELSE
           P     = Undefined
           A     = Undefined
           Ecc   = Undefined
           Inc   = Undefined
           Omega= Undefined
           Argp  = Undefined
           Nu    = Undefined
           M     = Undefined
           U     = Undefined
           l     = Undefined
           CapPi= Undefined
         ENDIF
      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                           SUBROUTINE RANDV
*
*  This Subroutine finds the position and velocity vectors in Geocentric
*    Equatorial (IJK) system given the classical orbit elements.  NOTICE
*    P is used for calculations and that semi major axis, a, is not.  This
*    convention allows parabolic orbits to be treated as well as the other
*    conic sections.  Notice the special cases leave Argp, Omega and Nu equal
*    to zero, rather than setting them to some large number as a flag for
*    infinite or undefined.  This allows the routine to process different types
*    of orbits with ONE transformation matrix since zeros will leave the vectors
*    unchanged during that phase of the transformation.
*
*  Algorithm     : Select the type of orbit through IF statements
*                     and assign Omega, Argp, and Nu
*                     Although these values change, they are NOT passed back
*                  Find the PQW position and veocity vectors
*                  Rotate by 3-1-3 to IJK.  Order is important
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs        :
*    P           - Semi-latus rectum                        DU
*    E           - eccentricity                             0.0 to ...
*    Inc         - inclination                              0.0 to Pi  rad
*    Omega       - Longitude of Ascending Node              0.0 to 2Pi rad
*    Argp        - Argument of Perigee                      0.0 to 2Pi rad
*    Nu          - True anomaly                             0.0 to 2Pi rad
*    U           - Argument of Latitude      (CI)           0.0 to 2Pi rad
*    L           - True Longitude            (CE)           0.0 to 2Pi rad
*    CapPi       - Longitude of Periapsis    (EE)           0.0 to 2Pi rad
*
*  Outputs       :
*    R           - IJK Position vector                      DU
*    V           - IJK Velocity vector                      DU / TU
*
*  Locals        :
*    Temp        - Temporary REAL value
*    Small       - Tolerance for roundoff errors
*    Rpqw        - PQW Position vector                      DU
*    Vpqw        - PQW Velocity vector                      DU / TU
*    TempVec     - PQW Velocity vector
*
*  Constants     :
*    Pi                                                      3.14159265358979
*
*  Coupling      :
*    MAG           Magnitude of a vector
*    ROT3          Rotation about the 3rd axis
*    ROT1          Rotation about the 1st axis
*
*  References    :
*    BMW           pg. 71-73, 80-83
*    Escobal       pg. 68-83
*
* -----------------------------------------------------------------------------
*
```

```
      SUBROUTINE RandV ( P,E,Inc,Omega,Argp,Nu,U,L,CapPi,  R,V )
        IMPLICIT NONE
        REAL*8 P,E,Inc,Omega,Argp,Nu,U,L,CapPi,R(4),V(4)

* ------------------------------ Locals ------------------------------
        REAL*8 Temp, Small,Rpqw(4), Vpqw(4), TempVec(4), Pi

* ------------------------ Initialize Variables ----------------------
        Small = 0.000001D0
        Pi    = 3.14159265358979D0

* --------------------------------------------------------------------
*  Determine what type of orbit is involved and set up the
*  set up angles for the special cases.
* --------------------------------------------------------------------
        IF ( E.LT.Small ) THEN
* ---------------------- Circular Equatorial --------------------------
            IF (( Inc.LT.Small ).or.( ABS(Inc - Pi).LT.Small )) THEN
                Argp = 0.0D0
                Omega= 0.0D0
                Nu   = L
* ---------------------- Circular Inclined  ---------------------------
              ELSE
                Argp= 0.0D0
                Nu  = U
              ENDIF
          ELSE
* --------------------- Elliptical Equatorial -------------------------
            IF (( Inc.LT.Small ).or.( ABS(Inc - Pi).LT.Small )) THEN
                Argp = Cappi
                Omega= 0.0D0
              ENDIF
          ENDIF

* -------------- Form PQW position and velocity vectors --------------
        Temp= P / (1.0D0 + E*DCOS(NU))
        Rpqw(1)= Temp*DCOS(NU)
        Rpqw(2)= Temp*DSIN(NU)
        Rpqw(3)=     0.0D0
        Vpqw(1)= -DSIN(NU)/DSQRT(P)
        Vpqw(2)=  (E + DCOS(NU)) / DSQRT(P)
        Vpqw(3)=      0.0D0
        CALL MAG( Rpqw )
        CALL MAG( Vpqw )

* ------------------ Perform transformation to IJK ------------------
        CALL ROT3( Rpqw   , -Argp , TempVec )
        CALL ROT1( TempVec, -Inc  , TempVec )
        CALL ROT3( TempVec, -Omega,  R      )

        CALL ROT3( Vpqw   , -Argp , TempVec )
        CALL ROT1( TempVec, -Inc  , TempVec )
        CALL ROT3( TempVec, -Omega,  V      )
      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                          SUBROUTINE GIBBS
*
*  This Subroutine performs the Gibbs method of orbit determination.  This
*    method determines the velocity at the middle point of the 3 given position
*    vectors.  Several flags are passed back.
*
*                  Flt = 0 ok
*                  Flt = 1 not coplanar
*                  Flt = 2 orbit impossible
*
*  The Gibbs method is best suited for coplanar, sequential position vectors
*    which are more than about 10 deg apart.  Notice the angle is passed back
*    so the user may make a decision about the accuracy of the calculations as
*    vectors which are 120 deg apart may be accurate, while vectors 8 deg
*    apart would not.  The method will calculate the resulting velocity using
*    the vectors IN THE ORDER GIVEN.  IF the calculations are not possible,
*    V2 is set to 0.0.  Notice a 1 deg tolerance is allowed for the coplanar
*    check.  This is necessary to allow for noisy data in the estimation project.
*
*  Algorithm     : Initialize values including the answer
*                  Find if the vectors are coplanar, else set a flag
*                  Check that the orbit is possible, else set a flag
*                  Find the largest angle between the vectors
*                  Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   28 Mar 1990
*
*  Inputs        :
*    R1          - IJK Position vector #1                DU
*    R2          - IJK Position vector #2                DU
*    R3          - IJK Position vector #3                DU
*
*  Outputs       :
*    V2          - IJK Velocity Vector for R2            DU / TU
*    Theta       - Angle between the two vectors         rad
*    Flt         - Flag indicating success               0, 1, 2
*
*  Locals        :
*    tover2      -
*    l           -
*    Small       - Tolerance for roundoff errors
*    r1mr2       - Magnitude of r1 - r2
*    r3mr1       - Magnitude of r3 - r1
*    r2mr3       - Magnitude of r2 - r3
*    p           - P Vector     r2 x r3
*    q           - Q Vector     r3 x r1
*    w           - W Vector     r1 x r2
*    d           - D Vector     p + q + w
*    n           - N Vector (r1)p + (r2)q + (r3)w
*    s           - S Vector (r2-r3)r1+(r3-r1)r2+(r1-r2)r3
*    b           - B Vector     d x r2
*    Theta1      - temporary angle between the two vectors rad
*    pN          - P Unit Vector
*    R1N         - R1 Unit Vector
*    dn          - D Unit Vector
*    nn          - N Unit Vector
*    i           - index
*
*  Constants     :
*    None.
*
*  Coupling      :
*    MAG           Magnitude of a vector
*    CROSS         Cross product of two vectors
*    DOT           Dot product of two vectors
*    ADVEC3        Add three vectors
*    LNCOM2        Multiply two vectors by two constants
*    LNCOM3        Add three vectors each multiplied by a constant
*    NORM          Creates a Unit Vector
*    ANGLE         Angle between two vectors
*
*  References    :
*    BMW           pg. 109-116
*    Escobal       pg. 306-307
*
* ------------------------------------------------------------------------
*
```

```
      SUBROUTINE GIBBS ( R1,R2,R3,  V2,Theta,flt )
        IMPLICIT NONE
        REAL*8   R1(4),R2(4),R3(4),V2(4),Theta
        INTEGER  Flt
        EXTERNAL DOT

* ---------------------------- Locals -----------------------------
        REAL*8 tover2, l, Small, r1mr2, r3mr1, r2mr3,p(4), q(4), w(4),
     &        d(4), n(4), s(4), b(4), Dot, PN(4), R1N(4),dn(4),nn(4),
     &        Theta1
        INTEGER i

* ------------------------ Initialize Variables ----------------------
        Small= 0.000001D0
        Flt  = 0
        Theta= 0.0D0
        CALL MAG( R1 )
        CALL MAG( R2 )
        CALL MAG( R3 )
        DO i= 1,4
            V2(i)= 0.0D0
        ENDDO

* ------------------------------------------------------------------
*  Determine if the vectors are coplanar.  The DOT product of R1 and the
*    normal vector of R2 and R3 will be 0 if all three vectors are coplanar.
*    The Vectors are normalized to accept very small and very large
*    vectors.  The magnitudes are left out of the DOT product equation :
*    r1n dot pn = r1n pn Cos() : since each vector is normalized, so the
*    magnitudes are 1.0.  A 1 deg tolerance is allowed for estimation, and
*    is implemented by allowing the angle between R1n and Pn to range from
*    89.0 to 91.0 deg, or Cos(89.0) = 0.017452406.
* ------------------------------------------------------------------
        CALL CROSS( R2,R3,P )
        CALL CROSS( R3,R1,Q )
        CALL CROSS( R1,R2,W )
        CALL NORM( P,PN )
        CALL NORM( R1, R1N )
        IF ( DABS( DOT(R1N,PN) ).GT.0.017452406D0 ) THEN
            Flt= 1
          ELSE
            CALL ADVEC3( P,Q,W,D )
            CALL LNCOM3( R1(4),R2(4),R3(4),P,Q,W,N )
            CALL NORM( N, NN )
            CALL NORM( D, DN )

* ------------------------------------------------------------------
*  Determine if the orbit is possible.  Both D and N must be in
*    the same direction, and non-zero.
* ------------------------------------------------------------------
            IF ( ( DABS(d(4)).LE.Small ).or.( DABS(n(4)).LE.SMALL )
     &                 .or.(DOT(nn,dn).LE.Small) ) THEN
                Flt= 2
              ELSE
                CALL ANGLE( R1,R2,  Theta )
                CALL ANGLE( R2,R3,  Theta1 )
                IF ( Theta1.GT.Theta ) THEN
                    Theta = Theta1
                  ENDIF

* ----------------- Perform Gibbs method to find V2  ----------------
                R1mr2= R1(4)-R2(4)
                R3mr1= R3(4)-R1(4)
                R2mr3= R2(4)-R3(4)
                CALL LNCOM3(R1mr2,R3mr1,R2mr3,R3,R2,R1,S)
                CALL CROSS( d,r2,b )
                L     = 1.0D0 / DSQRT(d(4)*n(4))
                Tover2= L/R2(4)
                CALL LNCOM2(Tover2,L,B,S,V2)

              ENDIF
          ENDIF
      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                           SUBROUTINE HERRGIBBS
*
*  This Subroutine implements the Herrick-Gibbs approximation for orbit
*    determination, and finds the middle velocity vector for the 3 given
*    position vectors.  The method is good for fast calculations and small
*    angles, <= 10 deg.  Notice the angle is passed back since vectors which
*    are 12 deg apart may actually be accurate, while vectors which are 170 deg
*    apart would not.  The observations MUST be sequential and taken on one
*    revolution.  The Use of Julian Dates for input makes it much easier to
*    perform calculations where the sights occur around midnight.  Several
*    flags are passed back:
*
*           Flt = 0 ok
*           Flt = 1 orbits not coplanar
*           Flt = 2 angles between the vectors are larger than 10 deg
*
*    Notice a 1 deg tolerance is allowed for the coplanar check.  This is
*    necessary to allow for noisy data in the estimation project.
*
*  Algorithm     : Initialize values including the answer
*                  Find if the vectors are coplanar, else set a flag
*                  Find the largest angle between the vectors
*                  Calculate the Taylor series for the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  28 Mar 1990
*
*  Inputs        :
*    R1          - IJK Position vector #1                DU
*    R2          - IJK Position vector #2                DU
*    R3          - IJK Position vector #3                DU
*    JD1         - Julian Date of 1st sighting           days from 4713 B.C.
*    JD2         - Julian Date of 2nd sighting           days from 4713 B.C.
*    JD3         - Julian Date of 3rd sighting           days from 4713 B.C.
*
*  Outputs       :
*    V2          - IJK Velocity Vector for R2            DU / TU
*    Theta       - Angle between the two vectors         rad
*    Flt         - Flag indicating success               0, 1, 2
*
*  Locals        :
*    dt21        - time delta between r1 and r2          TU
*    dt31        - time delta between r3 and r1          TU
*    dt32        - time delta between r3 and r2          TU
*    TolAngle    - Tolerance angle  (10 deg)             rad
*    Theta1      - temporary angle between the two vectorsrad
*    p           - P vector    r2 x r3
*    pN          - P Unit Vector
*    R1N         - R1 Unit Vector
*    Term1       - First Term for HGibbs expansion
*    Term2       - Second Term for HGibbs expansion
*    Term3       - Third Term for HGibbs expansion
*    i           - Index
*
*  Constants     :
*    TUMin       - Minutes in each Time Unit             13.44685108204
*
*  Coupling      :
*    MAG           Magnitude of a vector
*    CROSS         Cross product of two vectors
*    DOT           Dot product of two vectors
*    NORM          Creates a Unit Vector
*    LNCOM3        Combination of three vectors and three scalars
*    ANGLE         Find the angle between two vectors
*
*  References    :
*    Escobal       pg. 254-256, 304-306
*
* ------------------------------------------------------------------------
*
```

```
      SUBROUTINE HerrGibbs ( R1,R2,R3,JD1,JD2,JD3,  V2,Theta,Flt )
        IMPLICIT NONE
        REAL*8 R1(4),R2(4),R3(4),JD1,JD2,JD3,V2(4),Theta
        INTEGER Flt
        EXTERNAL DOT

* ----------------------------- Locals -----------------------------
        REAL*8 dt21, dt31, dt32, TolAngle,p(4), Theta1,
     &         TUMin, Dot, PN(4), R1N(4), Term1,Term2,Term3
        INTEGER i

* ------------------------ Initialize Variables ----------------------
        TUMin  = 13.44685108204D0
        Flt    =  0
        Theta  =  0.0D0
        CALL MAG( R1 )
        CALL MAG( R2 )
        CALL MAG( R3 )
        DO i= 1,4
            V2(i)= 0.0D0
        ENDDO
        TolAngle= 0.174532925D0
        DT21= (JD2-JD1)*1440.0D0/TUMin
        DT31= (JD3-JD1)*1440.0D0/TUMin
        DT32= (JD3-JD2)*1440.0D0/TUMin

* ------------------------------------------------------------------
*  Determine if the vectors are coplanar.  The DOT product of R1 and the
*    normal vector of R2 and R3 will be 0 if all three vectors are coplanar.
*    The Vectors are normalized to accept very small and very large
*    vectors.  The magnitudes are left out of the DOT product equation :
*    r1n dot pn = r1n pn Cos() : since each vector is normalized, so the
*    magnitudes are 1.0.  A 1 deg tolerance is allowed for estimation, and
*    is implemented by allowing the angle between R1n and Pn to range from
*    89.0 to 91.0 deg, or Cos(89.0) = 0.017452406.
* ------------------------------------------------------------------
        CALL CROSS( R2,R3,P )
        CALL NORM( P,PN )
        CALL NORM( R1, R1N )
        IF ( DABS( DOT(R1N,PN) ).GT.0.017452406D0 ) THEN
            Flt= 1
          ELSE

* ------------------------------------------------------------------
*  Check the size of the angles between the three position vectors.
*    Herrick Gibbs only gives "reasonable" answers when the
*    position vectors are reasonably close.  10 deg is only an estimate.
* ------------------------------------------------------------------
            CALL ANGLE( R1,R2,  Theta )
            CALL ANGLE( R2,R3,  Theta1 )
            IF ( Theta1.GT.Theta ) THEN
                Theta = Theta1
              ENDIF
            IF ( Theta.GT.TolAngle ) THEN
                Flt= 2
              ENDIF

* -------------- Perform Herrick-Gibbs method to find V2 --------------
            Term1 = -dt32*( 1.0D0/ (dt21*dt31) + 1.0D0/ (12*r1(4)**3) )
            Term2 = (dt32-dt21)*( 1.0D0/ (dt21*dt32) +
     &                      1.0D0/ (12*r2(4)**3) )
            Term3 =  dt21*( 1.0D0/ (dt32*dt31) + 1.0D0/ (12*r3(4)**3) )
            CALL LNCOM3( Term1,Term2,Term3,R1,R2,R3,  V2 )

          ENDIF
      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                            SUBROUTINE FINDCandS
*
*  This Subroutine calculates the C and S functions for use in the Universal
*    Variable calculations.   NOTE equality is handled by the series expansion
*    terms to eliminate potential discontinuities.  The series is only used for
*    negative values of Z since the truncation results in rather large errors
*    as Z gets larger than about 10.0.
*
*  Algorithm     : If Z is greater than zero, use the exact formulae else
*                    use the series form
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  30 Jan 1991
*
*  Inputs        :
*    ZNew        - Z variable
*
*  Outputs       :
*    CNew        - C function value
*    SNew        - S function value
*
*  Locals        :
*    ZSqrd       - ZNew squared
*    ZFrth       - ZNew to the fourth power
*    SqrtZ       - Square root of ZNew
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    BMW          pg. 207-210   (Complete graph of S and C)
*    Kaplan       pg. 304-305
*
* ---------------------------------------------------------------------------

      SUBROUTINE FindCandS ( ZNew,  CNew,SNew )
        IMPLICIT NONE
        REAL*8 ZNew,CNew,SNew

* ----------------------------  Locals  -----------------------------
        REAL*8 SqrtZ, ZSqrd, ZFrth

* ------------------------  Implementation   ------------------------
        IF ( ZNew.GT.0.0D0 ) THEN
            SqrtZ = SQRT( Znew )
            CNew  = (1.0D0-DCOS( SqrtZ )) / ZNew
            SNew  = (SqrtZ-DSIN( SqrtZ )) / ( SqrtZ**3 )
          ELSE
            ZSqrd = ZNew**2
            ZFrth = ZSqrd**2
            CNew  = 0.5D0 - ZNew/24.0D0  + ZSqrd/720.0D0
     &                - (ZSqrd*ZNew)/40320.0D0 + ZFrth/3628800.0D0
     &                - (ZFrth*ZNew)/479001600.0D0
            SNew  = 1.0D0/6.0D0 - ZNew/120.0D0 + ZSqrd/5040.0D0
     &                - (ZSqrd*ZNew)/362880.0D0 + ZFrth/39916800.0D0
     &                - (ZFrth*ZNew)/6227020800.0D0
          ENDIF
      RETURN
      END

*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE NEWTONR
*
*  This Subroutine performs the Newton Rhapson iteration to find the
*    Eccentric Anomaly given the Mean anomaly.  The True Anomaly is also
*    calculated.
*
*  Algorithm     : Setup the first guess
*                  Loop while the answer has not converged
*                  Write an error if the answer doesn't converge
*                  Find the True Anomaly using ATAN2 to resolve quadrants
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
*
*  Inputs        :
*    e           - Eccentricity                         0.0 - 1.0
*    M           - Mean Anomaly                         0.0 - 2Pi rad
*
*  Outputs       :
*    E0          - Eccentric Anomaly                    0.0 - 2Pi rad
*    Nu          - True Anomaly                         0.0 - 2Pi rad
*
*  Locals        :
*    E1          - Eccentric Anomaly, next value        rad
*    Sinv        - Sine of Nu
*    Cosv        - Cosine of Nu
*    i           - Index
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    BMW           pg. 184-186, 220-222
*
* ---------------------------------------------------------------------------

      SUBROUTINE NewtonR ( E,M,  E0,Nu )
        IMPLICIT NONE
        REAL*8 E,M,E0,Nu
        INTEGER i

* ----------------------------- Locals ------------------------------
        REAL*8 Sinv, Cosv, E1

* ------------------------ Initialize Variables -----------------------
        E0= M
        i= 1

* ------------- Newton Iteration for Eccentric Anomaly ---------------
        E1= E0 - ( ( E0 - e*DSIN(E0)-m ) / ( 1.0 - e*DCOS(E0) ) )

        DO WHILE ( (DABS(E1-E0).GT.0.0000001D0).and.(i.le.20) )
            E0= E1
            E1= E0 - ( ( E0 - e*DSIN(E0)-m ) / ( 1.0D0 - e*DCOS(E0) ) )
            i = i + 1
          ENDDO

        IF ( i.gt.20 ) THEN
            WRITE(*,*) 'Newton Rhapson not converged in 20 Iterations'
          ENDIF

* ------------------ Find True Anomaly at Epoch ---------------------
        Sinv= ( DSQRT( 1.0D0-e*e ) * DSIN(E1) ) / ( 1.0D0-e*DCOS(E1) )
        Cosv= ( DCOS(E1)-e ) / ( 1.0D0 - e*DCOS(E1) )
        NU  = DATAN2( Sinv,Cosv )
      RETURN
      END
*
```

```
* ----------------------------------------------------------------------------
*
*                           SUBROUTINE KEPLER
*
*  This Subroutine solves Keplers problem for orbit determination and returns a
*    future Geocentric Equatorial (IJK) position and velocity vector.  The
*    solution Subroutine uses Universal variables.
*
*  Algorithm     : Initialize variables
*                  Find size and shape parameters for all cases
*                  Setup initial guesses with IF statements
*                  Loop while the time has not converged
*                  If too many iterations, print an error
*                     otherwise calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
*
*  Inputs        :
*    Ro          - IJK Position vector - initial          DU
*    Vo          - IJK Velocity vector - initial          DU / TU
*    Time        - Length of time to propagate            TU
*
*  Outputs       :
*    R           - IJK Position vector                    DU
*    V           - IJK Velocity vector                    DU / TU
*
*  Locals        :
*    F           - f expression
*    G           - g expression
*    FDot        - f dot expression
*    GDot        - g dot expression
*    XOld        - Old Universal Variable X
*    XOldSqrd    - XOld squared
*    XNew        - New Universal Variable X
*    XNewSqrd    - XNew squared
*    ZNew        - New value of z
*    CNew        - C(z) function
*    SNew        - S(z) function
*    DeltaT      - change in t                            TU
*    TimeNew     - New time                               TU
*    RDotV       - Result of Ro dot Vo
*    A           - Semi major axis                        DU
*    Alpha       - Reciprocol  1/a
*    SME         - Specific Mech Energy                   DU2 / TU2
*    Period      - Time period for satellite TU
*    S           - Variable for parabolic case
*    W           - Variable for parabolic case
*    Temp        - Temporary real value
*    Small       - Tolerance for roundoff errors
*    i           - index
*
*  Constants     :
*    HalfPi                                               1.57079632679490
*    TwoPi                                                6.28318530717959
*    Infinite    - Flag for an Infinite element           999999.9
*
*  Coupling      :
*    MAG           Magnitude of a vector
*    DOT           Dot product of two vectors
*    COT           Cotangent function
*    FindCandS     Find C and S functions
*
*  References    :
*    Kaplan        pg. 304-308  ( Includes first guess for x if parabolic)
*    BMW           pg. 191-199, 203-212
*
* ----------------------------------------------------------------------------
*
```

```
      SUBROUTINE Kepler ( Ro,Vo,Time,  R,V )
        IMPLICIT NONE
        REAL*8 Ro(4),Vo(4),Time,R(4),V(4)
        EXTERNAL DOT, COT

* ----------------------------- Locals -----------------------------
        REAL*8 F, G, FDot, GDot, DeltaT, XOld,XOldSqrd,XNew,XNewSqrd,
     &       ZNew, CNew, SNew,TimeNew,RDotV,A,Alpha,
     &       SME,Period,S,W,Temp, Small,TwoPi,HalfPi,Dot, Cot,Infinite
        INTEGER i

* ------------------------ Initialize Variables ----------------------
        HalfPi   = 1.57079632679490D0
        TwoPi    = 6.28318530717959D0
        Infinite= 999999.9
        Small    = 0.000001D0
        TimeNew = -10.0D0
        CALL MAG( Ro )
        CALL MAG( Vo )
        RDotV= DOT( Ro,Vo )
        DO i= 1,4
            V(i)= 0.0D0
        ENDDO

* ---------------------- Find SME, Alpha, and A ----------------------
        SME= ( Vo(4)**2 / 2.0D0 ) - ( 1.0D0/Ro(4) )
        Alpha= -SME*2.0D0
        IF (DABS( SME ).GT.Small) THEN
            A= -1.0D0 / ( 2.0D0*SME )
          ELSE
            A= Infinite
          ENDIF
        IF (DABS( Alpha ).LT.Small) THEN
            Alpha= 0.0D0
          ENDIF

* --------------------   Setup initial guess for x  ------------------
* ----------------------- Circle and Ellipse ------------------------
        IF (Alpha.GE.Small) THEN
            Period= TwoPi * DSQRT( DABS(A)**3 )
            IF (DABS( Time ).GT.ABS( Period )) THEN
                Time= DMOD( Time,Period )
              ENDIF
            IF (DABS(Alpha-1.0D0).GT.Small) THEN
                 XOld = Time * Alpha
              ELSE
* ------ Make sure 1st guess isn't too close for a circle, r=a --------
                 XOld= Time*Alpha*0.97D0
              ENDIF
          ELSE
* ------------------------------ Parabola ----------------------------
            IF (DABS( Alpha ).LT.Small) THEN
                S= 0.5D0 * (HalfPi - DATAN(
     &             3.0D0*DSQRT( 1.0D0/(Ro(4)**3) )* Time ) )
                W= DATAN( DTAN( S )**(1.0D0/3.0D0 ) )
                XOld = DSQRT(Ro(4))*( 2.0D0*COT(2.0D0*W) )
                Alpha= 0.0D0
              ELSE
* ----------------------------- Hyperbola ----------------------------
                  Temp= -2.0D0*Time /
     &                  ( A*( RDotV + SIGN(1.0D0,Time)*
     &                    DSQRT(-A)*(1.0D0-Ro(4)/a) ) )
                  XOld= SIGN( 1.0D0,Time ) * DSQRT( -A ) * DLOG( Temp )
              ENDIF
          ENDIF

*
```

```
      i= 1
      DO WHILE ( (DABS(TimeNew-Time).GT.0.000001D0).and.(i.LE.15) )
          XOldSqrd= XOld**2
          ZNew    = XOldSqrd * Alpha

* ---------------------- Find C and S functions ----------------------
          CALL FindCandS( ZNew,CNew,SNew )

* -------------- Use a Newton iteration for new values ---------------
          TimeNew = XOldSqrd*XOld*SNew + RDotV*XOldSqrd*CNew +
     &               Ro(4)*XOld*( 1.0D0 - ZNew*SNew )
          DeltaT  = XOldSqrd*CNew + RDotV*XOld*( 1.0D0 - ZNew*SNew )+
     &               Ro(4)*( 1.0D0 - ZNew*CNew )

* ------------------  Calculate new value for x  --------------------
          XNew = XOld + ( Time-TimeNew ) / DeltaT

* ------------------------------------------------------------------
* Check if the orbit is an ellipse and xnew.GT.2pi SQRT(a), the step
*   size must be changed.  This is accomplished by multiplying DeltaT
*   by 10.0.  NOTE !! 10.0 is arbitrary, but seems to produce good
*   results.  The idea is to keep XNew from increasing too rapidily.
* ------------------------------------------------------------------
          IF ( (A.GT.0.0D0 ).and.( ABS(XNew).GT.TwoPi*DSQRT(A)).and.
     &          (SME.LT.0.0D0) ) THEN
              XNew = XOld + ( Time-TimeNew ) / ( DeltaT*10.0D0 )
            ENDIF
*
*         Write( *,60 ) i,XOld,TimeNew,DeltaT,XNew,SNew,CNew,znew
*  60     FORMAT( I2,1X,7(F10.5) )
*
*
          i= i + 1
          XOld = XNew

        ENDDO

      IF ( i.GT.15 ) THEN
          Write (*,*) 'Kepler not converged in 15 iterations '
        ELSE

* ------  Calculate position and velocity vectors at new time ---------
          XNewSqrd = XNew**2
          F      = 1.0D0 - ( XNewSqrd*CNew / Ro(4) )
          G      = Time - XNewSqrd*XNew*SNew
          DO i= 1,3
              R(i)= F*Ro(i) + G*Vo(i)
          ENDDO
          CALL MAG( R )
          GDot  = 1.0D0 - ( XNewSqrd*CNew / R(4) )
          FDot  = ( XNew / ( Ro(4)*R(4) ) ) * ( ZNew*SNew - 1.0D0 )
          DO i= 1,3
              V(i)= FDot*Ro(i) + GDot*Vo(i)
          ENDDO
          CALL MAG( V )
        ENDIF

      RETURN
      END

*
```

```
* ------------------------------------------------------------------------------
*
*                           SUBROUTINE GAUSS
*
*  This Subroutine solves the Gauss problem of orbit determination and returns
*    the velocity vectors at each of two given position vectors.  The solution
*    uses Universal Variables for calculation and a bissection technique for
*    updating Z.  This method is slower than the Newton iteration discussed in
*    BMW, but it does NOT suffer problems with negative z values, and is valid
*    for ellipses LESS THAN one revolution, parabolas, and Hyperbolas.  Also
*    note the selection of small since the algorithm is very sensitive to
*    changes in this variable.  A value of 0.001 will converge in say 10
*    iterations instead of 25 iterations with 0.000001, and the accuracy will
*    differ in the 3rd-4th decimal place.  I chose to keep the higher accuracy
*    for cases like example 13, BMW pg. 274, #5.10.
*    ( Refer to graph on BMW pg. 235 for ranges of z. )
*
*  Algorithm     : Initialize variables and setup initial guesses
*                  Loop while the time has not converged
*                  If too many iterations, print an error
*                     otherwise calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs        :
*    R1          - IJK Position vector 1                    DU
*    R2          - IJK Position vector 2                    DU
*    DM          - direction of motion                      'L','S'
*    Time        - Time between R1 and R2                   TU
*
*  Outputs       :
*    V1          - IJK Velocity vector                      DU / TU
*    V2          - IJK Velocity vector                      DU / TU
*
*  Locals        :
*    VarA        - Variable of the iteration, NOT the semi major axis!
*    Y           -
*    Upper       - Upper bound for Z
*    Lower       - Lower bound for Z
*    CosDeltaNu  - Cosine of true anomaly change            rad
*    F           - f expression
*    G           - g expression
*    GDot        - g dot expression
*    XOld        - Old Universal Variable X
*    XOldCubed   - XOld cubed
*    ZOld        - New value of z
*    ZNew        - New value of z
*    CNew        - C(z) function
*    SNew        - S(z) function
*    TimeNew     - New time                                 TU
*    Small       - Tolerance for roundoff errors
*    i           - index
*    j           - index
*
*  Constants     :
*    TwoPi                                                    6.28318530717959
*
*  Coupling      :
*    MAG           Magnitude of a vector
*    DOT           Dot product of two vectors
*    FindCandS     Find C and S functions
*
*  References    :
*    BMW           pg. 228-241  (Uses a Newton iteration)
*
* ------------------------------------------------------------------------------
*
```

```
      SUBROUTINE GAUSS ( R1,R2,DM,Time,  V1,V2 )
        IMPLICIT NONE
        REAL*8 R1(4),R2(4),Time,V1(4),V2(4)
        CHARACTER DM
        EXTERNAL DOT

* ------------------------------ Locals ------------------------------
        REAL*8 VarA, Y, Upper, Lower, CosDeltaNu, F, G, GDot,
     &         XOld, XOldCubed,ZOld, ZNew, CNew, SNew, TimeNew,
     &         Small,TwoPi, Dot
        INTEGER i, j

* ------------------------ Initialize Variables ----------------------
        TwoPi  = 6.28318530717959D0
        Small  = 0.000001D0
        TimeNew= -10.0D0
        CALL MAG( R1 )
        CALL MAG( R2 )
        DO i= 1,4
            V1(i)=   ^D0
            V2(i)= 0.0D0
        ENDDO
        CosDeltaNu= DOT(R1,R2)/(R1(4)*R2(4))
        IF (Dm.EQ.'L') THEN
            VarA   = -DSQRT( R1(4)*R2(4)*(1.0D0+CosDeltaNu) )
          ELSE
            VarA   =  DSQRT( R1(4)*R2(4)*(1.0D0+CosDeltaNu) )
          ENDIF

* --------------------- Form Initial guesses   -----------------------
        ZOld= 0.0D0
        CNew = 0.5D0
        SNew = 1.0D0/6.0D0
* ------------------    Bounds for Z iteration  ----------------------
        Upper= TwoPi**2
        Lower= -2.0D0*TwoPi

* ------------ Determine if the orbit is possible at all -------------
        IF (DABS( VarA ).GT.Small) THEN

* --------------------------------------------------------------------
*  Perform Gaussian Iteration using Universal Variables.  Notice
*   the iteration is performed using a bissection technique instead of
*   a Newton itertion.  Although the Newton iteration is quicker, the
*   bissection will not fail with large negative Z values.  The upper
*   and lower bounds are adjusted as required to keep y from becoming -.
* --------------------------------------------------------------------

*
```

```
      i= 0
      DO WHILE ( (DABS(TimeNew-Time).GT.Small).and.(i.LE.30) )
          Y= R1(4)+R2(4)-( VarA*(1.0D0-ZOld*SNew)/DSQRT(CNew) )

* ------------------------------------------------------------------
*  A check is needed for special cases where VarA is greater than 0.0.
*   It's possible that Z can become very negative, and cause the square
*   root in the XOld calculation to blow up.  This section loops until
*   the ZNew value will result in a + y value.  The solution is to slowly
*   update the lower bound of Z until y is +.  The 0.8* for ZNew is simply
*   a means to let Z change a little slower.  The ZNew equation is found
*   by solving the y equation for z when y = 0.
* ------------------------------------------------------------------
          IF (( VarA.GT.0.0D0 ).and.( Y.LT.0.0D0 )) THEN
              j= 1
             DO WHILE ( ( Y.LT.0.0D0 ).and.( j.LT.10 ) )
                 ZNew= 0.8D0*(1.0D0/SNew)*( 1.0D0 -
     &                     ( R1(4)+R2(4))*DSQRT(CNew)/VarA   )

* ---------------------- Find C and S functions ----------------------
                 CALL FindCandS( ZNew, CNew,SNew )
                 ZOld= ZNew
                 Lower= ZOld
                 Y= R1(4) + R2(4) -
     &                ( VarA*(1.0D0-ZOld*SNew)/DSQRT(CNew) )
                 j = j + 1
               ENDDO
            IF (j.GE.10) THEN
                 WRITE(*,*) 'Iteration failed for Yn in Gauss'
               ENDIF
            ENDIF
          XOld= DSQRT( Y/CNew )
          XOldCubed= XOld**3
          TimeNew = XOldCubed*SNew + VarA*DSQRT(Y)

* --------------------  Readjust upper and lower bounds   -------------
          IF (TimeNew.LT.Time) THEN
              Lower= ZOld
            ENDIF
          IF (TimeNew.GT.Time) THEN
              Upper= ZOld
            ENDIF
          ZNew= ( Upper+Lower ) / 2.0D0
*
*        Write( *,60 ) i,ZOld,Y,XOld,TimeNew,VarA,upper,lower
*  60    FORMAT( I2,1X,7(F10.5) )
*
* -----------  Make sure the first guess isn't too close --------------
          IF ((DABS(TimeNew-Time).LT.Small).and.(i.EQ.0)) THEN
               TimeNew = -10.0D0
             ENDIF

* --------------------- Find C and S functions -----------------------
          CALL FindCandS( ZNew, CNew,SNew )
          ZOld = ZNew
          i= i + 1

        ENDDO

      IF ( i.GE.30 ) THEN
          Write (*,*) 'Gauss not converged in 30 iterations '
        ELSE

* -----------  Use F and G series to find Velocity Vectors   ----------
          F    = 1.0D0 - ( Y / R1(4) )
          G    = VarA*DSQRT( Y )
          GDot = 1.0D0 - Y/R2(4)
          DO i= 1,3
              V1(i)= ( R2(i) - F*R1(i) )/G
              V2(i)= ( GDot*R2(i) - R1(i) )/G
          ENDDO
          CALL MAG( V1 )
          CALL MAG( V2 )
        ENDIF

    ELSE
      Write( *,* ) 'Gauss problem cannot be solved'
    ENDIF
  RETURN
  END
*
```

```
* ----------------------------------------------------------------------------
*
*                       SUBROUTINE IJKtoLATLON
*
* This Subroutine converts a Geocentric Equatorial (IJK) position vector into
*   latitude and longitude.  Geodetic and Geocentric latitude are found.
*
* Algorithm     : Initialize variables
*                 Find the longitude being careful to reslove the angle
*                 Setup iteration for latitude
*                 Loop while the deltas are not equal
*                 Write an error message if the values do not converge
*
* Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
* Inputs        :
*   R           - IJK position vector                    DU
*   JD          - Julian Date                            days from 4713 B.C.
*
* Outputs       :
*   GeoCnLat    - Geocentric Latitude                    -Pi/2 to Pi/2 rad
*   Lon         - Longitude (WEST -)                     -2Pi to 2Pi rad
*
* Locals        :
*   Rc          - Range of site w.r.t. earth center      DU
*   Height      - Height above earth w.r.t. site         DU
*   Alpha       - Angle from I axis to point, LST        rad
*   OldDelta    - Previous value of DeltaLat             rad
*   DeltaLat    - Diff between Delta and Geocentric lat  rad
*   GeoDtLat    - Geodetic Latitude                      rad
*   TwoFMinusF2 - 2*F - F squared
*   OneMinusF2  - ( 1 - F ) squared
*   Delta       - Declination angle of R in IJK system   rad
*   RSqrd       - Magnitude squared                      DU2
*   Temp        - Diff between Geocentric/Geodetic lat   rad
*   GST         - Greenwich Sidereal Time                rad
*   SinTemp     - Sine of Temp                           rad
*   i           - index
*
* Constants     :
*   Pi          -                                         3.14159265358979
*   TwoPi       -                                         6.28318530717959
*   Flat        - Flatenning of the Earth                 0.003352810664747352
*
* Coupling      :
*   MAG           Magnitude of a vector
*   GSTime        Greenwich Sidereal Time
*
* References    :
*   Escobal       pg. 398-399
*
* ----------------------------------------------------------------------------
*
```

```
      SUBROUTINE IJKtoLatLon ( R, JD, GeoCnLat, Lon )
        IMPLICIT NONE
        REAL*8 R(4),JD,GeoCnLat,Lon
        EXTERNAL GSTime

* ------------------------------- Locals -------------------------------
        REAL*8 Rc, Height, Alpha, OldDelta, DeltaLat, GeoDtLat,
     &         TwoFMinusF2,OneMinusF2, Delta, Temp, GST, RSqrd,
     &         Pi,TwoPi,Flat, GSTime, SinTemp
        INTEGER i

* ------------------------- Initialize values  -----------------------
        Pi     = 3.14159265358979D0
        TwoPi  = 6.28318530717959D0
        Flat   = 0.003352810664747352D0
        TwoFMinusF2 = 2.0D0*Flat - Flat**2
        OneMinusF2  = ( 1.0D0-Flat )**2
        CALL MAG( R )

* -------------------- Find Longitude value --------------------------
        Temp = DSQRT( R(1)*R(1) + R(2)*R(2) )
        Alpha = DATan2( R(2) / Temp , R(1) / Temp )
        GST = GSTIME( JD )
        Lon = Alpha - GST
        IF ( DABS(Lon).GE.Pi ) THEN
         IF ( Lon.LT.0.0 ) THEN
             Lon = TwoPi + Lon
           ELSE
             Lon = Lon - TwoPi
           ENDIF
         ENDIF

* ------------------ Set up initial latitude value  ------------------
        Delta  = DATan( R(3) / Temp )
        IF ( DABS(Delta).GT.Pi ) THEN
            Delta = DMOD( Delta,Pi )
          ENDIF
        GeoCnLat = Delta
        OldDelta =  1.0D0
        DeltaLat = 10.0D0
        RSqrd    = R(4)**2

* -------- Iterate to find Geocentric and Geodetic Latitude  ----------
        i = 1
        DO WHILE ( ( DABS(OldDelta-DeltaLat).GT.0.00001D0).and.
     &          (i.LT.10) )
            OldDelta = DeltaLat
            Rc       = DSQRT( ( 1.0D0-TwoFMinusF2 ) /
     &                 ( 1.0D0-TwoFMinusF2*DCOS(GeoCnLat)**2 ) )
            GeoDtLat = DATan( DTAN(GeoCnLat) / OneMinusF2 )
            Temp     = GeoDtLat-GeoCnLat
            SinTemp  = DSIN(Temp)
            Height   = DSQRT( RSqrd - Rc**2*SinTemp**2 ) -
     &                        Rc*DCOS(Temp)
            DeltaLat = DASIN( Height*SinTemp / R(4) )
            GeoCnLat = Delta - DeltaLat
            i = i + 1
          ENDDO

        IF ( i.GE.10 ) THEN
            Write(*,*) 'IJKtoLatLon did NOT converge '
          ENDIF
      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                          SUBROUTINE SUN
*
*  This Subroutine calculates the Geocentric Equatorial position vector for
*    the Sun given the Julian Date.  This is the low precision formula and
*    is valid for years from 1950 to 2050.  Accuaracy of apparent coordinates
*    is 0.01 degrees.  Notice many of the calculations are performed in
*    degrees, and are not changed until later.  This is due to the fact that
*    the Almanac uses degrees exclusively in their formulations.
*
*  Algorithm     : Calculate the several values needed to find the vector
*                  Be careful of quadrant checks
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  25 Aug 1988
*
*  Inputs        :
*    JD          - Julian Date                               days from 4713 B.C.
*
*  Outputs       :
*    RSun        - IJK Position vector of the Sun            AU
*    RtAsc       - Right Ascension                           rad
*    Decl        - Declination                               rad
*
*  Locals        :
*    MeanLong    - Mean Longitude
*    MeanAnomaly - Mean anomaly
*    N           - Number of days from 1 Jan 2000
*    EclpLong    - Ecliptic longitude
*    Obliquity   - Mean Obliquity of the Ecliptic
*
*  Constants     :
*    Pi                                                       3.14159265358979
*    TwoPi                                                    6.28318530717959
*    Rad                                                      57.29577951308230
*
*  Coupling      :
*    None.
*
*  References    :
*    1987 Astronomical Almanac Pg. C24
*
* ---------------------------------------------------------------------------
*
```

```
      SUBROUTINE Sun ( JD,  RSun,RtAsc,Decl )
        IMPLICIT NONE
        REAL*8 JD,RSun(4),RtAsc,Decl

* ------------------------------ Locals ------------------------------
        REAL*8 MeanLong, MeanAnomaly, EclpLong, Obliquity, N, Pi,
     &         TwoPi, Rad

* ------------------------- Initialize values -----------------------
        Pi    =  3.14159265358979D0
        TwoPi=  6.28318530717959D0
        Rad   = 57.29577951308230D0
        N     = ( JD - 2451545.0D0 )

        MeanLong= 280.460D0 + 0.9856474D0*N
        MeanLong= DMOD( MeanLong,360.0D0 )

        MeanAnomaly= 357.528D0 + 0.9856003D0*N
        MeanAnomaly= DMOD( MeanAnomaly/Rad,TwoPi )
        IF (MeanAnomaly.LT.0.0D0) THEN
            MeanAnomaly= TwoPi + MeanAnomaly
          ENDIF

        EclpLong = MeanLong + 1.915D0*DSIN(MeanAnomaly) +
     &             0.020D0*DSIN(2.0D0*MeanAnomaly)
        Obliquity= 23.439D0 - 0.0000004D0*N

        MeanLong    = MeanLong/Rad
        IF (MeanLong.LT.0.0D0) THEN
            MeanLong= TwoPi + MeanLong
          ENDIF
        EclpLong    = EclpLong / Rad
        Obliquity   = Obliquity / Rad

        RtAsc= DATAN( DCOS(Obliquity)*DTAN(EclpLong) )

* ---------- Check that RtAsc is in the same quadrant as EclpLong -----
* ------------------ make sure it's in 0 to 2pi range   ---------------
        IF (EclpLong.LT.0.0D0) THEN
            EclpLong= EclpLong + TwoPi
          ENDIF
        IF (DABS(EclpLong-RtAsc).GT.(Pi/2.0D0)) THEN
            RtAsc= RtAsc + 0.5D0*Pi*DNINT((EclpLong-RtAsc)/(0.5D0*Pi))
          ENDIF

        Decl = DASIN( DSIN(Obliquity)*DSIN(EclpLong) )

* ---------- Find magnitude of SUN vector, then components ------------
        RSun(4)= 1.00014D0 - 0.01671D0*DCOS( MeanAnomaly )
     &                     - 0.00014D0*DCOS( 2.0D0*MeanAnomaly )
        RSun(1)= RSun(4)*DCOS( EclpLong )
        RSun(2)= RSun(4)*DCOS(Obliquity)*DSIN(EclpLong)
        RSun(3)= RSun(4)*DSIN(Obliquity)*DSIN(EclpLong)

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------
*
*                          SUBROUTINE MOON
*
*  This subroutine calculates the Geocentric Equatorial (IJK) position vector
*    for the moon given the Julian Date.  This is the low precision formula and
*    is valid for years between 1950 and 2050.  Notice many of the calculations
*    are performed in degrees.  This coincides with the development in the
*    Almanac.  The equation for Ecliptic Longitude was split in two to prevent
*    software problems with numeric coprocessors.  The error seemed to be a
*    stack overflow since the equation is so long.  The program errors are as
*    follows:
*
*                Eclpitic Longitude  0.3   degrees
*                Eclpitic Latitude   0.2   degrees
*                Horiz Parallax      0.003 degrees
*                Distance from Earth 0.2   DUs
*                Right Ascension     0.3   degrees
*                Declination         0.2   degrees
*
*  Algorithm     : Find the initial quantities
*                  Calculate direction cosines
*                  Find the position and velocity vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  25 Aug 1988
*
*  Inputs        :
*    JD          - Julian Date                            days from 4713 B.C.
*
*  Outputs       :
*    RMoon       - IJK Position vector of the Moon        DU
*    RtAsc       - Right Ascension                        rad
*    Decl        - Declination                            rad
*
*  Locals        :
*    EclpLong    - Ecliptic Longitude
*    EclpLat     - Eclpitic Latitude
*    HzParal     - Horizontal Parallax
*    l           - Geocentric Direction Cosines
*    m           -             "     "
*    n           -             "     "
*    Tu          - Julian Centuries from 1 Jan 1900
*    x           - Temporary REAL value
*
*  Constants     :
*    TwoPi                                                 6.28318530717959
*    Rad                                                   57.29577951308230
*
*  Coupling      :
*    None.
*
*  References    :
*    1987 Astronomical Almanac Pg. D46
*    Explanatory Supplement( 1960 ) pg. 106-111
*    Roy, Orbital Motion Pg. 61-62 ( Discussion of parallaxes )
*
* -----------------------------------------------------------------------
*
```

```
      SUBROUTINE Moon ( JD,RMoon,RtAsc,Decl )
        IMPLICIT NONE
        REAL*8 JD,RMoon(4),RtAsc,Decl

* ------------------------------ Locals ------------------------------
        REAL*8 EclpLong, EclpLat, HzParal, l,m,n,Tu,TwoPi, Rad, x

* ------------------------- Initialize values -----------------------
        TwoPi=  6.28318530717959D0
        Rad  =  57.29577951308230D0
        Tu   =  ( JD - 2451545.0D0 ) / 36525.0D0

        x       =     218.32D0 + 481267.883D0*Tu
     &                + 6.29*DSin( (134.9D0+477198.85D0*Tu)/Rad )
     &                - 1.27*DSin( (259.2D0-413335.38D0*Tu)/Rad )
     &                + 0.66*DSin( (235.7D0+890534.23D0*Tu)/Rad )

        EclpLong= x + 0.21D0*DSin( (269.9D0+954397.70D0*Tu)/Rad )
     &                - 0.19D0*DSin( (357.5D0+ 35999.05D0*Tu)/Rad )
     &                - 0.11D0*DSin( (186.6D0+966404.05D0*Tu)/Rad )

        EclpLat =       5.13D0*DSin( ( 93.3D0+483202.03D0*Tu)/Rad )
     &                + 0.28D0*DSin( (228.2D0+960400.87D0*Tu)/Rad )
     &                - 0.28D0*DSin( (318.3D0+  6003.18D0*Tu)/Rad )
     &                - 0.17D0*DSin( (217.6D0-407332.20D0*Tu)/Rad )

        x       =       0.9508D0 +
     &                  0.0518D0*DCos( (134.9+477198.85*Tu)/Rad )

        HzParal = x + 0.0095D0*DCos( (259.2D0-413335.38D0*Tu)/Rad )
     &                + 0.0078D0*DCos( (235.7D0+890534.23D0*Tu)/Rad )
     &                + 0.0028D0*DCos( (269.9D0+954397.70D0*Tu)/Rad )

        EclpLong   = DMOD( EclpLong/Rad, TwoPi )
        EclpLat    = DMOD( EclpLat/Rad, TwoPi )
        HzParal    = DMOD( HzParal/Rad, TwoPi )

* --------------- Find the geocentric direction cosines ---------------
        l= DCOS( EclpLat ) * DCOS( EclpLong )
        m= 0.9175D0*DCOS(EclpLat)*DSIN(EclpLong)
     &                - 0.3978D0*DSIN(EclpLat)
        n= 0.3978D0*DCOS(EclpLat)*DSIN(EclpLong)
     &                + 0.9175D0*DSIN(EclpLat)

* ---------------- Calculate Moon position vector --------------------
        RMoon(4)= 1.0D0/DSIN( HzParal )
        RMoon(1)= RMoon(4)*l
        RMoon(2)= RMoon(4)*m
        RMoon(3)= RMoon(4)*n

* ----------------- Find Rt Ascension and Declination ----------------
        RtAsc= DATan2( m,l )
        Decl = DASIN( n )

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------
*
*                           SUBROUTINE PLANETRV
*
*  This subroutine calculates the planetary ephemerides using the Epoch J2000.
*    The coefficients are obtained from Danbys book and provisions are left
*    to obtain Heliocentric Equatorial, or Heliocentric Ecliptic coordinates.
*    Notice the ephemeris presents data wrt the solar equator.
*
*  Algorithm     : Use a case statement to assign each planets values
*                    Find the vectors
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  19 Dec 1989
*
*  Inputs        :
*    NumPlanet   - Number of planet                         1..9
*    JD          - Julian Date                              days from 4713 B.C.
*
*  Outputs       :
*    R           - XYZ position vector                      AU
*    V           - XYZ velocity vector                      AU / TU
*
*  Locals        :
*    u           -
*    l           -
*    cappi       -
*    TU          -
*    N           -
*    obliquity   -
*    a           -
*    e           -
*    p           -
*    inc         -
*    omega       -
*    argp        -
*    nu          -
*    m           -
*    LLong       -
*    LongP       -
*    e0          -
*
*  Coupling      :
*    NewtonR
*    RandV
*
*  Constants     :
*    TwoPi
*
*  References    :
*    Danby          pg.
*    Escobal        pg. 261-270
*
* -----------------------------------------------------------------------

      SUBROUTINE PlanetRV ( NumPlanet,JD,  R,V )
        IMPLICIT NONE

        REAL*8 R(4),V(4),JD
        INTEGER NumPlanet

* ---------------------------- Locals -----------------------------
        REAL*8  TUDAySun,u,l,cappi,Tu,n,obliquity,
     &          TwoPi,a,e,p,inc,omega,argp,nu,llong,longp,m,e0,Rad
        INTEGER i

* ------------------------ Implementation  ------------------------
        TwoPi   =  6.28318530717959D0
        Rad     = 57.29577951308230D0

        Tu  = ( JD - 2451545.0D0 ) / 36525.0D0
*
```

```
* ------------------------------ Mercury ----------------------------
      IF (NumPlanet.eq.1) THEN
          LongP=     1.3518643 +   0.0271656*Tu + 0.000005166*Tu*Tu
          Omega=     0.8435332 +   0.0207029*Tu + 0.000003072*Tu*Tu
          Inc  =     0.1222601 +   0.0000318*Tu - 0.000000314*Tu*Tu
          e    =     0.2056318 +   0.0000204*Tu - 0.000000030*Tu*Tu
          LLong=     4.4026098 +2608.8147071*Tu + 0.000005306*Tu*Tu
          a    =     0.3871035
        ENDIF
* ------------------------------ Venus   ----------------------------
      IF (NumPlanet.eq.2) THEN
          LongP=     2.2962199 +   0.0244734*Tu - 0.000018727*Tu*Tu
     &                                     - 0.000000087*Tu*Tu*Tu
          Omega=     1.3383171 +   0.0157275*Tu + 0.000007103*Tu*Tu
          Inc  =     0.0592480 +   0.0000175*Tu - 0.000000017*Tu*Tu
          e    =     0.0067719 -   0.0000478*Tu
          LLong=     3.1761467 +1021.3529430*Tu + 0.000005428*Tu*Tu
          a    =     0.7233074
        ENDIF
* ------------------------------ Earth   ----------------------------
      IF (NumPlanet.eq.3) THEN
          LongP=     1.7965956 +   0.0300116*Tu + 0.000008029*Tu*Tu
          Omega=     0.0000000
          Inc  =     0.0000000
          e    =     0.0167086 -   0.0000420*Tu
          LLong=    17.4614336 + 628.3319667*Tu + 0.000005306*Tu*Tu
          a    =     1.0000116
        ENDIF
* ------------------------------ Mars    ----------------------------
      IF (NumPlanet.eq.4) THEN
          LongP=     5.8653576 +   0.0321323*Tu + 0.000000236*Tu*Tu
          Omega=     0.8649519 +   0.0134756*Tu + 0.000000279*Tu*Tu
          Inc  =     0.0322838 -   0.0000105*Tu + 0.000000227*Tu*Tu
          e    =     0.0934006 +   0.0000905*Tu - 0.000000080*Tu*Tu
          LLong=     6.2034809 + 334.0856279*Tu + 0.000005428*Tu*Tu
          a    =     1.5237107
        ENDIF
* ------------------------------ Jupiter ----------------------------
      IF (NumPlanet.eq.5) THEN
          LongP=     6.5333138 +   0.0281458*Tu + 0.000017994*Tu*Tu
     &                                     - 0.000000070*Tu*Tu*Tu
          Omega=     1.7534353 +   0.0178190*Tu + 0.000006999*Tu*Tu
          Inc  =     0.0227464 -   0.0000959*Tu + 0.000000087*Tu*Tu
          e    =     0.0484949 +   0.0001632*Tu - 0.000000470*Tu*Tu
          LLong=     0.5995465 +  52.9934808*Tu + 0.000003910*Tu*Tu
          a    =     5.2102156
        ENDIF
* ------------------------------ Saturn  ----------------------------
      IF (NumPlanet.eq.6) THEN
          LongP=     1.6241473 +   0.0342741*Tu + 0.000014626*Tu*Tu
     &                                     + 0.000000087*Tu*Tu*Tu
          Omega=     1.9838376 +   0.0153082*Tu - 0.000002112*Tu*Tu
     &                                     - 0.000000035*Tu*Tu*Tu
          Inc  =     0.0434391 -   0.0000652*Tu - 0.000000262*Tu*Tu
          e    =     0.0555086 -   0.0003468*Tu - 0.000001000*Tu*Tu
          LLong=     0.8740168 +  21.3542956*Tu + 0.000009076*Tu*Tu
          a    =     9.5380701
        ENDIF
* ------------------------------ Uranus  ----------------------------
      IF (NumPlanet.eq.7) THEN
          LongP=     3.0195096 +   0.0259422*Tu + 0.000003752*Tu*Tu
          Omega=     1.2916474 +   0.0090954*Tu + 0.000023387*Tu*Tu
     &                                     + 0.000000332*Tu*Tu*Tu
          Inc  =     0.0134948 +   0.0000135*Tu + 0.000000646*Tu*Tu
          e    =     0.0462959 -   0.0000273*Tu + 0.000000080*Tu*Tu
          LLong=     5.4812939 +   7.5025431*Tu + 0.000005306*Tu*Tu
          a    =    19.1833020
        ENDIF
* ------------------------------ Neptune ----------------------------
      IF (NumPlanet.eq.8) THEN
          LongP=     0.8399169 +   0.0248931*Tu + 0.000006615*Tu*Tu
          Omega=     2.3000657 +   0.0192371*Tu + 0.000004538*Tu*Tu
          Inc  =     0.0308915 -   0.0001625*Tu - 0.000000140*Tu*Tu
          e    =     0.0089881 +   0.0000064*Tu
          LLong=     5.3118863 +   3.8376877*Tu + 0.000005393*Tu*Tu
          a    =    30.0551440
        ENDIF
* ------------------------------ Pluto   ----------------------------
      IF (NumPlanet.eq.9) THEN
          LongP=     3.9202678
          Omega=     1.9269569
          Inc  =     0.2990156
          e    =     0.2508770
          LLong=     3.8203049
          a    =    39.5375800
        ENDIF
```

```
        LLong= DMOD( LLong ,TwoPI )
        LongP= DMOD( LongP ,TwoPI )
        Omega= DMOD( Omega ,TwoPI )

        Argp= LongP - Omega
        M   = LLong - LongP

        CALL NewTonR( e,M,  E0,Nu )
        p= a*(1.0D0-e*e)

        u    = 0.0D0
        l    = 0.0D0
        Cappi= 0.0D0

        CALL RANDV( P,e,Inc,Omega,Argp,Nu,U,L,CapPi, R,V )
*
*  Alternate method for finding position vector
*     r(4)= ( a*( 1.0-e*e) ) / ( 1.0+e*Cos(Nu) )
*     r(1)= r(4)*( Cos(Nu+Argp)*Cos(Omega)-Sin(Nu+Argp)*Cos(Inc)*Sin(Omega) )
*     r(2)= r(4)*( Cos(Nu+Argp)*Sin(Omega)+Sin(Nu+Argp)*Cos(Inc)*Cos(Omega) )
*     r(3)= r(4)*Sin(Nu+Argp)*Sin(Inc)


* --------- Calculations required for reference to mean equator -------
        N    = ( JD - 2451545.0D0 )
        Obliquity = (23.439 - 0.0000004D0*N) / Rad

        CALL ROT1( R ,-Obliquity, R )
        CALL ROT1( V ,-Obliquity, V )

        TUDaySun= 54.20765355D0
        DO 10 i= 1, 3
           v(i)= v(i)/tudaysun
 10       CONTINUE

*     IF (Show.eq.'Y') THEN
*        Write(*,*) '         a          e          i       Omega',
*     &              '     LongP  ')
*        Write(*,5) a,E,Inc*rad,Omega*rad,LongP*rad
*        Write(*,*) '    LLong         Argp          M            Nu'
*        Write(*,6) LLong*rad,Argp*rad,M*rad,Nu*rad
*        Write(*,*) 'JD = ',JD
*      ENDIF
*  5   FORMAT( 5(F12.7,1X) )
*  6   FORMAT( 4(F12.7,1X) )

      RETURN
      END
*
```

```
* ----------------------------------------------------------------------------
*
*                              FUNCTION GEOCENTRIC
*
*  This Function converts from Geodetic to Geocentric latitude.  Notice that
*    (1-f) squared = 1-eSqrd.
*
*  Algorithm     : Find the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
*
*  Inputs        :
*    Lat         - Geodetic Latitude                        -Pi/2 - Pi/2 rad
*
*  Outputs       :
*    Geocentric  - Geocentric Latitude                      -Pi/2 - Pi/2 rad
*
*  Locals        :
*    None.
*
*  Constants     :
*    EESqrd      - Eccentricity of Earth squared         0.00669437999013
*
*  Coupling      :
*    None.
*
*  References    :
*    Escobal       pg. 136
*    Kaplan        pg. 332-336
*
* ----------------------------------------------------------------------------

      REAL*8 FUNCTION Geocentric ( Lat )
        IMPLICIT NONE
        REAL*8 Lat

        REAL*8 EESqrd
* ------------------------  Initialize values   ----------------------
        EESqrd     = 0.00669437999013D0
        Geocentric= DATAN( (1.0D0 - EESqrd)*DTAN(Lat) )
      RETURN
      END
* ----------------------------------------------------------------------------
*
*                              FUNCTION INVGEOCENTRIC
*
*  This Function converts from Geocentric to Geodetic latitude.  Notice that
*    (1-f) squared = 1-eSquared.
*
*  Algorithm     : Find the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  12 Aug 1988
*
*  Inputs        :
*    Lat         - Geocentric Latitude                      -Pi/2 - Pi/2 rad
*
*  Outputs       :
*    InvGeocentric- Geodetic Latitude                        -Pi/2 - Pi/2 rad
*
*  Locals        :
*    None.
*
*  Constants     :
*    EESqrd      - Eccentricity of Earth squared         0.00669437999013
*
*  Coupling      :
*    None.
*
*  References    :
*    Escobal       pg. 136
*    Kaplan        pg. 332-336
*
* ----------------------------------------------------------------------------

      REAL*8 FUNCTION InvGeocentric ( Lat )
        IMPLICIT NONE
        REAL*8 Lat

        REAL*8 EESqrd
* ------------------------  Initialize values   ----------------------
        EESqrd     = 0.00669437999013D0
        InvGeocentric= DATAN( DTAN(Lat)/(1.0D0 - EESqrd) )
      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                             SUBROUTINE SIGHT
*
*  This subroutine takes the position vectors of two satellites and determines
*    if there is line-of-sight between the two satellites.  A spherical Earth
*    with radius of 1 DU is assumed.  The process is to form the equation of
*    a line between the two vectors.  Differentiating and setting to zero finds
*    the minimum value, and when plugged back into the original line equation,
*    gives the minimum distance.  The parameter tmin is allowed to range from
*    0.0 to 1.0.
*
*  Algorithm     : Find tmin
*                  Check value of tmin for LOS
*                  Find dist squared if needed
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  31 Jan 1990
*
*  Inputs        :
*    R1          - Position vector of the first sat        DU
*    R2          - Position vector of the second sat       DU
*
*  Outputs       :
*    LOS         - Line of Sight                            'Yes','No '
*
*  Locals        :
*    ADotB       - Dot product of a dot b
*    THin        - Minimum value of t from a to b
*    DistSqrd    - Distance squared for min dist to earth DU
*    ASqrd       - Magnitude of A squared
*    BSqrd       - Magnitude of B squared
*
*  Constants     :
*    None.
*
*  Coupling      :
*    DOT           Dot product of two vectors
*
*  References    :
*    None.
*
* ---------------------------------------------------------------------------

      SUBROUTINE SIGHT         ( R1,R2,          LOS        )
        IMPLICIT NONE
        REAL*8 R1(4), R2(4)
        CHARACTER*3 LOS
        EXTERNAL DOT

* ------------------------------ Locals ------------------------------
        REAL*8 Dot,ADotB, THin,DistSqrd,ASqrd,BSqrd

        BSqrd = R2(4)**2
        ASqrd = R1(4)**2
        ADotB = DOT( R1,R2 )
        THin = ( ASqrd - ADotB ) / ( ASqrd + BSqrd - 2.0D0*ADotB )

        IF ( (THin.lt.0.0D0).or.(THin.gt.1.0D0) ) THEN
            LOS = 'YES'
          ELSE
            DistSqrd = (1.0D0-THin)*ASqrd + ADotB*THin
            IF (DistSqrd.gt.1.0D0) THEN
                LOS = 'YES'
              ELSE
                LOS = 'NO '
              ENDIF
          ENDIF

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------------
*
*                              SUBROUTINE LIGHT
*
*  This subroutine determines if a spacecraft is sunlit or in the dark at a
*    particular time.  A spherical Earth and cylindrical shadow is assumed.
*
*  Algorithm     : Find the sun vector
*                  Use the sight algorithm for the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   9 Feb 1990
*
*  Inputs        :
*    R           - IJK Position vector of satellite       DU
*    JD          - Julian Date of desired observation     days
*
*  Outputs       :
*    Vis         - Visibility Flag                        'Yes','No '
*
*  Locals        :
*    RtAsc       - Suns Right ascension                   rad
*    Decl        - Suns Declination                       rad
*    RSun        - Sun vector                             AU
*    AUDU        - Conve sion from AU to DU
*
*  Constants     :
*    None
*
*  Coupling      :
*    SUN           Position vector of Sun
*    LNCOM1        Multiple a vector by a constant
*    SIGHT         Does Line-of-sight exist bewteen vectors
*
*  References    :
*    Escobal       pg.
*
* ------------------------------------------------------------------------------

      SUBROUTINE LIGHT         ( R,JD,           VIS       )
        IMPLICIT NONE
        REAL*8 R(4),JD
        Character*3 Vis

* -----------------------------  Locals  ------------------------------
        REAL*8 RSun(4),AUDU,RtAsc,Decl

* -------------------------  Implementation  -------------------------
        AUDU = 149599650.0D0/6378.137D0

        CALL SUN( JD,RSun,RtAsc,Decl )
        CALL LNCOM1( AUDU,RSun, RSun )

*        Write(*,10) 'RSun =',RSun(1),RSun(2),RSun(3),RSun(4)
* 10     FORMAT( A5,4(F14.8) )

* -------------- Is the satellite in the shadow or not  ---------------
        CALL SIGHT( RSun,R,  Vis )

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                            SUBROUTINE OMS2
*
*  This subroutine determines the velocity and position vector of the shuttle
*    after it performs the OMS-2 burn.  Assume the burn and the resulting
*    velocity change are instanteanous.
*
*  Algorithm     : Find the velocity vector
*                  Rotate to IJK
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   7 Mar 1990
*
*  Inputs        :
*    Lat         - Geodetic latitude of the shuttle's Earth sub-
*                       point (its NADAR) before the burn.          rad
*    Lon         - Geodetic longitude of the shuttle's NADAR        rad
*    Alt         - Altitude of the shuttle above the Earth's surface  DU
*    Phi         - Shuttle flight path angle                        rad
*    Az          - Shuttle azimuth angle                            rad
*    Speed       - Shuttle scalar velocity with respect to inertial space DU/TU
*    JD          - Julian Date                                   Ref 4713 B.C.
*
*  Outputs :
*    R           - Position vector of the shuttle after the OMS2 burn DU
*    V           - Inertial velocity vector of the shuttle after OMS2 burn DU/TU
*
*  Locals        :
*    VSEZ        - Velocity vector expressed in the SEZ frame       DU/TU
*
*  Constants     :
*    HalfPi
*
*  Coupling      :
*    LSTIME      - Find LST and GST
*    SITE        - Find Site vector on an oblate Earth
*    ROT2        - Rotate about the 2 axis
*    ROT3        - Rotate about the 3 axis
*
*  References    :
*    None.
*
* ---------------------------------------------------------------------------

      SUBROUTINE OMS2( Lat,Lon,Alt,Phi,Az,Speed,JD,   R,V )
        IMPLICIT NONE
        REAL*8 Lat,Lon,Alt,Phi,Az,Speed,R(4),V(4),JD

* ------------------------- Local Variables -------------------------
        REAL*8 GST, LST, VSEZ(4),VS(4),HalfPi,TempVec(4)

* ------------------------ Initialize Variables -----------------------
        HalfPi     = 1.57079632679490D0

        CALL LSTime( Lon,JD, Lst,Gst )
        CALL SITE( Lat,Alt,Lst, R,VS )

* ------ Velocity vector in the rotating, Earth-fixed SEZ frame -------
        VSEZ(1) = -Speed * DCOS(Phi) * DCOS(Az)
        VSEZ(2) =  Speed * DCOS(Phi) * DSIN(Az)
        VSEZ(3) =  Speed * DSIN(Phi)
        CALL MAG( VSEZ )

* ---------------- Perform SEZ to IJK transformation -----------------
        CALL ROT2( VSEZ,    Lat-HalfPi, TempVec )
        CALL ROT3( TempVec,   -LST    , V       )

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------
*
*                            SUBROUTINE RNGAZ
*
*  This subroutine calculates the Range and Azimuth between two specified
*    ground points on a spherical Earth.  Notice the range will ALWAYS be
*    within the range of values listed since you do not know the direction of
*    firing, long or short.  The procedure will calculate Rotating Earth ranges
*    if the TOF is passed in other than 0.0.
*
*  Algorithm     : Find the range
*                  Calculate the Az noting all combinations of quadrants
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  25 Aug 1988
*
*  Inputs        :
*    LLat        - Start Geocentric Latitude           -Pi/2 - Pi/2 rad
*    LLon        - Start Longitude (WEST -)             0.0 - 2Pi rad
*    TLat        - End Geocentric Latitude             -Pi/2 - Pi/2 rad
*    TLon        - End Longitude (WEST -)               0.0 - 2Pi rad
*    TOF         - Time of flight if ICBM, or 0.0       TU
*
*  Outputs       :
*    Range       - Range between points                 0.0 -  Pi rad
*    Az          - Azimuth                              0.0 - 2Pi rad
*
*  Locals        :
*    Small       - Tolerance
*
*  Constants     :
*    TwoPi                                               6.28318530717959
*    Pi                                                  3.14159265358979
*    OmegaEarth  - Angular rotation of Earth (Rad/TU)   0.0588335906868878
*
*  Coupling      :
*    None.
*
*  References    :
*    BMW           pg. 309-311
*
* -----------------------------------------------------------------------

      SUBROUTINE RngAz ( LLat,LLon,TLat,TLon,TOF,  Range,Az )
        IMPLICIT NONE
        REAL*8 LLat,LLon,TLat,TLon,TOF,Range,Az

* ----------------------------- Locals ------------------------------
        REAL*8 Small, Pi, TwoPi, OmegaEarth

* ------------------------ Initialize values  -----------------------
        Pi         =  3.14159265358979D0
        Small      =  0.000001D0
        OmegaEarth=  0.0588335906868878D0
        TwoPi      =  6.28318530717959D0

        Range= DACOS( DSIN(LLat)*DSIN(TLat) + DCOS(LLat)*DCOS(TLat)*
     &                DCOS(TLon-LLon + OmegaEarth*TOF) )

* ---------- Check if range is 0 or half the Earth distance -----------
        IF ( DABS( DSin(Range)*DCos(LLat) ).LT. Small) THEN
           IF ( DABS( Range - Pi ).LT.Small ) THEN
               Az= Pi
             ELSE
               Az= 0.0D0
             ENDIF
         ELSE
           Az= DACOS( ( DSIN(TLat) - DCOS(Range) * DSIN(LLat)) /
     &                    ( DSIN(Range) * DCOS(LLat)) )
         ENDIF

* ---------- Check if the Azimuth is grt than 180 degrees -------------
        IF (DSIN( TLon - LLon + OmegaEarth*TOF ).LT.0.0D0) THEN
            Az= TwoPi - Az
          ENDIF

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE PATH
*
* This subroutine determines the end position for a given range and azimuth
*   from a given point.  Notice the use of ATAN2 to eliminate quadrant
*   problems.  Also, Geocentric coordinates are used since the Earth is
*   assumed to be spherical.
*
* Algorithm     : Find the latitude
*                 Find the change in longitude noting quadrant possibilities
*                 Calculate the longitude
*
* Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  25 Aug 1988
*
* Inputs        :
*   LLat        - Start Geocentric Latitude            -Pi/2 - Pi/2 rad
*   LLon        - Start Longitude                       0.0 - 2Pi rad
*   Range       - Range between points                  DU
*   Az          - Azimuth                               0.0 - 2Pi rad
*
* Outputs       :
*   TLat        - End Geocentric Latitude              -Pi/2 - Pi/2 rad
*   TLon        - End Longitude                         0.0 - 2Pi rad
*
* Locals        :
*   SinDeltaN   - Sine of Delta N                       rad
*   CosDeltaN   - Cosine of Delta N                     rad
*   DeltaN      - Angle bteween the two points          rad
*   Small       - Tolerance
*
* Constants     :
*   Pi                                                   3.14159265358979
*   TwoPi                                                6.28318530717959
*
* Coupling      :
*   None.
*
* References    :
*   BMW           pg. 309-311
*
* ---------------------------------------------------------------------------
*
```

```
      SUBROUTINE Path ( LLat,LLon,Range,Az,  TLat,TLon )
        IMPLICIT NONE
        REAL*8 LLat,LLon,Range,Az,TLat,TLon

* ------------------------------ Locals ------------------------------
        REAL*8 SinDeltaN,CosDeltaN, DeltaN, Small, TwoPi, Pi

* ------------------------- Initialize values   -----------------------
        Pi    = 3.14159265358979D0
        TwoPi= 6.28318530717959D0
        Small= 0.0000001D0

        Az= DMOD( Az,TwoPi )
        IF (LLon.LT.0.0D0) THEN
            LLon= TwoPi + LLon
          ENDIF
        IF (Range.GT.TwoPi) THEN
            Range= DMOD( Range,TwoPi )
          ENDIF

* --------------------- Find Geocentric Latitude  ----------------------
        TLat = DASIN( DSIN(LLat)*DCOS(Range) +
     &                DCOS(LLat)*DSIN(Range)*DCOS(Az) )

* ------------- Find Delta N, the angle between the points ------------
        IF ((DABS(DCOS(TLat)).GT.Small).and.
     &      (DABS(DCOS(LLat)).GT.Small) ) THEN
            SinDeltaN= DSIN(Az)*DSIN(Range) / DCOS(TLat)
            CosDeltaN= ( DCOS(Range)-DSIN(TLat)*DSIN(LLat) ) /
     &                 ( DCOS(TLat)*DCOS(LLat) )
            DeltaN= DATan2(SinDeltaN,CosDeltaN)
          ELSE
* ------------ Case where launch is within 3nm of a Pole  -------------
            IF (DABS(DCOS(LLat)).LE.Small) THEN
                IF ((Range.GT.Pi).and.(Range.LT.TwoPi)) THEN
                    DeltaN= Az + Pi
                  ELSE
                  ENDIF
              ENDIF
* ----------- Case where end point is within 3nm of a pole ------------
            IF (DABS( DCOS(TLat) ).LE.Small) THEN
                DeltaN= 0.0D0
              ENDIF
          ENDIF

        TLon= LLon + DeltaN
        IF (TLon.LT.0.0D0) THEN
            TLon= TwoPi + TLon
          ENDIF
        IF (TLon.GT.TwoPi) THEN
            TLon= DMOD( TLon,TwoPi )
          ENDIF

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                          SUBROUTINE TRAJEC
*
*  This subroutine calculates the Range, Azimuth, and Time of Flight between
*    two specified ground points for an ICBM with a known Q.  Calculations
*    depend on knowledge of burnout conditions, and the iterations are
*    performed for either a high or low trajectory.  Notice the ICBM will fly
*    on an inertial trajectory, and values for earth relative velocities,
*    etc., are calculated after the iteration.  Notice these calculations do
*    not support trajectories over half the world away.
*
*  Algorithm     : Find the Range and Az with 0 TOF
*                  If the trajectory is possible,
*                    Loop to find the Range and Az corrected
*                    Calculate influence coefficients
*                    Find velocity needed
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   9 Oct 1988
*
*  Inputs        :
*    LLat        - Start Geocentric Latitude           -Pi/2 - Pi/2 rad
*    LLon        - Start Longitude (WEST -)             0.0 - 2Pi rad
*    TLat        - End Geocentric Latitude             -Pi/2 - Pi/2 rad
*    TLon        - End Longitude (WEST -)               0.0 - 2Pi rad
*    Rbo         - Radius at burnout                    DU
*    Q           - Non-dimensional Q performance based on Inertial Velocity
*    TypePhi     - Type of trajectory, High ot Low      'H', 'L'
*
*  Outputs       :
*    Range       - Rotating Range between points        0.0 -  Pi rad
*    Phi         - Inertial Flight Path Angle           rad
*    TOF         - Rotating Earth Time of Fligth        TU
*    Az          - Inert Azimuth                        0.0 - 2Pi rad
*    ICPhi       - Influence Coefficient for Phi        rad/rad
*    ICVbo       - Influence Coefficient for Vbo        rad/ du/tu
*    ICRbo       - Influence Coefficient for Rbo        rad/rad
*    Vn          - Velocity the missile needs to provide  DU/TU
*
*  Locals        :
*    Small       - Tolerance
*    QBoMin      - Minimum Q for a given range
*    VEarth      -
*    VBo         -
*    a           -
*    Ecc         -
*    E           -
*    RangeOld    -
*    i           - Index
*
*  Constants     :
*    Pi                                                  3.14159265358979
*    OmegaEarth  - Angular rotation of Earth (Rad/TU)    0.0588335906868878
*    Undefined   - Flag for an undefined element         999999.1
*
*  Coupling      :
*    MAG           Magnitude of a vector
*    RngAz         Finds the range and Azimuth between points
*
*  References    :
*    BMW           pg. 293-313
*
* ------------------------------------------------------------------------
*
```

```
      SUBROUTINE Trajec ( LLat,LLon,TLat,TLon,Rbo,Q,TypePhi,Range,
     &                    Phi,TOF,Az,ICPhi,ICVbo,ICRbo,Vn   )
        IMPLICIT NONE
        REAL*8 LLat,LLon,TLat,TLon,Rbo,Q,Range,Phi,TOF,Az,ICPhi,ICVbo,
     &         ICRbo,Vn(4)
        CHARACTER TypePhi
* ------------------------------ Locals ------------------------------
        REAL*8 a,Ecc,E,RangeOld,Vbo,VEarth,OmegaEarth,Small,Pi,
     &         QboMIn,Undefined
        INTEGER i
* ------------------------- Initialize values -----------------------
        OmegaEarth=  0.0588335906868878D0
        Small      =  0.000001D0
        Pi         =  3.14159265358979D0
        RangeOld   = -1.0D0
        Undefined  = 999999.1D0
        i          =  1

* ----------------- Iterate to find the flight time -----------------
        CALL RngAz( LLat,LLon,TLat,TLon,0.0D0, Range,Az )
        A      = RBo / (2.0D0 - Q)
        QBoMin= ( 2.0D0*DSin(Range/2.0D0) ) / (1.0D0+DSin(Range/2.0D0))
        IF (Q.GE.QboMin) THEN

            DO WHILE ((DABS(RangeOld-Range).GT.Small).and.( i.lt.20 ))
* ---------------- Check for High or Low Flight Path Angle ------------
                IF (TypePhi.EQ.'H') THEN
                    Phi= 0.5D0*( (Pi - DASIN(((2.0D0-Q)/Q)*
     &                            DSin(Range/2.0D0)) )- Range/2.0D0)
                  ELSE
                    Phi= 0.5D0*( DASIN(((2.0D0-Q)/Q)*DSin(Range/2.0D0))
     &                           - Range/2.0D0)
                  ENDIF
                Ecc = DSQRT( 1.0D0 + Q*(Q-2.0D0)*DCos(Phi)*DCos(Phi) )
                E   = DACos( (Ecc-DCos(Range/2.0D0)) / (1.0D0-Ecc*
     &                       DCos(Range/2.0D0)) )
                TOF = DSQRT(A**3)*2.0D0*( Pi - E + Ecc*DSIN(E) )

*               Write(*,*) i,Range*Rad,Phi*Rad,e*Rad,ecc,TOF*13.44685
                RangeOld = Range
                CALL RngAz( LLat,LLon,TLat,TLon,TOF, Range,Az )
                i= i+1
              ENDDO
            IF (i.GE.20 ) THEN
                Write(*,*) 'TRAJEC Not Converged in 20 Iteraions '
              ENDIF

* --------- Evaluate Influence Coefficients for unit errors -----------
            VBo   = DSQRT( Q/Rbo )
            ICPhi= ( ( 2.0D0*DSin(Range + 2.0D0*Phi) ) /
     &                             DSin(2.0D0*Phi) ) - 2.0D0
            ICVbo= ( 8.0D0*DSin(Range/2.0D0)*DSin(Range/2.0D0) ) /
     &                     ( Vbo**3*Rbo*DSin(2.0D0*Phi) )
            ICRbo= ( 4.0D0*DSin(Range/2.0D0)*DSin(Range/2.0D0) ) /
     &                     ( Vbo**2*RBo**2*DSin(2.0D0*Phi) )

* -------------- Find Velocity Needed, Relative Velocity --------------
            VEarth= OmegaEarth * DCos(LLat)

            VN(1)= -VBo*DCOS( Phi )*DCOS(Az)
            VN(2)=  VBo*DCOS( Phi )*DSIN(Az) - VEarth
            VN(3)=  VBo*DSIN( Phi )
            CALL MAG( VN )
          ELSE
            Write(*,*) 'The ICBM does not have enough energy  - '
            Write(*,*) 'Q Min = ',QboMin
            Phi   = Undefined
            TOF   = Undefined
            ICPhi= Undefined
            ICVbo= Undefined
            ICRbo= Undefined
            Vn(4)= Undefined
          ENDIF

      RETURN
      END
*
```

```
* ----------------------------------------------------------------------
*
*                          SUBROUTINE HOHMANN
*
*  This subroutine calculates the delta v's for a Hohmann transfer for either
*    circle to circle, or ellipse to ellipse.  The notation used is from the
*    initial orbit (1) at point a, transfer is made to the transfer orbit (2),
*    and to the final orbit (3) at point b.
*
*  Algorithm     : Find initial values
*                  If the orbits are both cir or ellip, find the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  19 Dec 1989
*
*  Inputs        :
*    R1          - Initial position magnitude           DU
*    R3          - Final position magnitude             DU
*    e1          - Eccentricity of first orbit
*    e3          - Eccentricity of final orbit
*    Nu1         - True Anomaly of first orbit          0 or Pi rad
*    Nu3         - True Anomaly of final orbit          0 or Pi rad
*
*  Outputs       :
*    DelVa       - Change in velocity at point a        DU / TU
*    DelVb       - Change in velocity at point b        DU / TU
*    TOF         - Time of Flight for the transfer      TU
*
*  Locals        :
*    SME1        - Specfic Mechanical Energy of first orbit     DU2 / TU
*    SME2        - Specfic Mechanical Energy of transfer orbit  DU2 / TU
*    SME3        - Specfic Mechanical Energy of final orbit     DU2 / TU
*    V1          - Velocity of 1st orbit at point a             DU / TU
*    V2a         - Velocity of transfer orbit at point a        DU / TU
*    V2b         - Velocity of transfer orbit at point b        DU / TU
*    V3          - Velocity of final orbit at point b           DU / TU
*    a1          - Semi Major Axis of first orbit               DU
*    a2          - Semi Major Axis of Transfer orbit            DU
*    a3          - Semi Major Axis of final orbit               DU
*
*  Constants     :
*    Pi                                                  3.14159265358979
*
*  Coupling      :
*    None.
*
*  References    :
*    BMW           pg. 163-166
*
* ----------------------------------------------------------------------
*
```

```
      SUBROUTINE Hohmann ( R1,R3,e1,e3,Nu1,Nu3,  DelVa,DelVb,TOF )
        IMPLICIT NONE

        REAL*8 R1,R3,e1,e3,Nu1,Nu3,DelVa,DelVb,TOF

* ------------------------------ Locals ------------------------------
        REAL*8 SME1,SME2,SME3, V1,V2a,V2b,V3, a1,a2,a3, Pi

* ----------------------- Initialize values  -----------------------
        Pi =  3.14159265358979D0
        a1  = (r1*(1.0D0+e1*DCos(Nu1))) / (1.0D0 - e1*e1 )
        a2  = ( R1 + R3 ) / 2.0
        a3  = (r3*(1.0D0+e3*DCos(Nu3))) / (1.0D0 - e3*e3 )
        SME1 = -1.0D0 / (2.0D0*a1)
        SME2 = -1.0D0 / (2.0D0*a2)
        SME3 = -1.0D0 / (2.0D0*a3)
        DelVa= 0.0D0
        DelVb= 0.0D0
        TOF= 0.0D0

        IF ( (e1.lt.1.0D0).or.(e3.lt.1.0D0) ) THEN

* -------------------- Find Delta v at point a --------------------
            V1  = DSQRT( 2.0D0*( (1.0D0/R1) + SME1 ) )
            V2a = DSQRT( 2.0D0*( (1.0D0/R1) + SME2 ) )
            DelVa= DABS( V2a - V1 )

* -------------------- Find Delta v at point b --------------------
            V3  = DSQRT( 2.0D0*( (1.0D0/R3) + SME3 ) )
            V2b = DSQRT( 2.0D0*( (1.0D0/R3) + SME2 ) )
            DelVb= DABS( V3 - V2b )

* ------------------- Find Transfer Time of Flight ----------------
            TOF= Pi * DSQRT( A2**3 )

*            IF (Show.eq.'Y') THEN
*                Write(*,*) ' a2 ',a2
*                Write(*,*) 'V1  ',v1,' V2a = ',v2a
*                Write(*,*) 'V2b ',v2b,' V3 = ',v3,' TOTAL',(DelVa+DelVb)
*              ENDIF

          ENDIF

      RETURN
      END
*
```

```
* --------------------------------------------------------------------------
*
*                            SUBROUTINE ONETANGENT
*
*  This subroutine calculates the delta v's for a One Tangent transfer for either
*    circle to circle, or ellipse to ellipse.  The notation used is from the
*    initial orbit (1) at point a, transfer is made to the transfer orbit (2),
*    and to the final orbit (3) at point b.
*
*  Algorithm     : Find the parameters for the transfer orbit
*                  Based on the eccentricity, find the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  19 Dec 1989
*
*  Inputs        :
*    R1          - Initial position magnitude            DU
*    R3          - Final position magnitude              DU
*    e1          - Eccentricity of first orbit
*    e3          - Eccentricity of final orbit
*    Nu1         - True Anomaly of first orbit           rad
*    Nu2         - True Anomaly of second orbit          rad
*    Nu3         - True Anomaly of final orbit           rad
*
*  Outputs       :
*    DelVa       - Change in velocity at point a          DU / TU
*    DelVb       - Change in velocity at point b          DU / TU
*    TOF         - Time of Flight for the transfer       TU
*
*  Locals        :
*    SME1        - Specfic Mechanical Energy of first orbit     DU2 / TU
*    SME2        - Specfic Mechanical Energy of transfer orbit  DU2 / TU
*    SME3        - Specfic Mechanical Energy of final orbit     DU2 / TU
*    V1          - Velocity of 1st orbit at point a             DU / TU
*    V2a         - Velocity of transfer orbit at point a        DU / TU
*    V2b         - Velocity of transfer orbit at point b        DU / TU
*    V3          - Velocity of final orbit at point b           DU / TU
*    e2          - Eccentricity of second orbit
*    a1          - Semi Major Axis of first orbit               DU
*    a2          - Semi Major Axis of Transfer orbit            DU
*    a3          - Semi Major Axis of final orbit               DU
*    E           - Eccentric anomaly of transfer orbit at b     rad
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    BMW           pg. 163-166
*
* --------------------------------------------------------------------------
*
```

```
      SUBROUTINE OneTangent ( R1,R3,e1,e3,Nu1,Nu2,Nu3,  DelVa,
     &                        DelVb,TOF )
        IMPLICIT NONE

        REAL*8 R1,R3,e1,e3,Nu1,Nu2,Nu3,DelVa,DelVb,TOF

* ------------------------------ Locals ------------------------------
        REAL*8 SME1,SME2,SME3, V1,V2a,V2b,V3, e2,a1,a2,a3, Phi2b,Phi3,
     &         E, Sinv,Cosv

* ------------------------ Initialize values ------------------------
        a1  = (r1*(1.0D0+e1*DCos(Nu1))) / (1.0D0 - e1*e1 )
        e2  = ( r3-r1 ) / ( -r3*DCos(Nu2)+DCos(Nu1)*r1 )
* Cos(Nu1) determines the sign
        IF ( DABS( e2-1.0D0 ) .gt. 0.0001D0 ) THEN
            a2   = (r1*(1.0D0+e2*DCos(Nu1))) / (1.0D0 - e2*e2 )
            SME2 = -1.0D0 / (2.0D0*a2)
          ELSE
            a2   = 999999.9D0
* Undefined for Parabolic orbit
            SME2 = 0.0D0
          ENDIF

        a3  = (r3*(1.0D0+e3*DCos(Nu3))) / (1.0D0 - e3*e3 )
        SME1 = -1.0D0 / (2.0D0*a1)
        SME3 = -1.0D0 / (2.0D0*a3)

* --------------------- Find Delta v at point a ---------------------
        V1  = DSQRT( 2.0D0*( (1.0/R1) + SME1 ) )
        IF ( DABS( SME2 ) .gt. 0.0001D0 ) THEN
            V2a = DSQRT( 2.0*( (1.0D0/R1) + SME2 ) )
          ELSE
            V2a = DSQRT( 2.0*(1.0D0/R1) )
          ENDIF
        DelVa= DABS( V2a - V1 )

* --------------------- Find Delta v at point b ---------------------
        V3  = DSQRT( 2.0D0*( (1.0D0/R3) + SME3 ) )
        IF ( DABS( SME2 ) .gt. 0.000001D0 ) THEN
            V2b = DSQRT( 2.0D0*( (1.0D0/R3) + SME2 ) )
          ELSE
            V2b = DSQRT( 2.0D0*(1.0D0/R3) )
          ENDIF

        Phi2b = DATAN( ( e2*DSin(Nu2) ) / ( 1.0D0 + e2*DCos(Nu2) ) )
        Phi3  = DATAN( ( e3*DSin(Nu3) ) / ( 1.0D0 + e3*DCos(Nu3) ) )
        DelVb= DSQRT( V2b*V2b + V3*V3 -
     &                2.0D0*V2b*V3*DCos( Phi2b-Phi3 ) )

* --------------------- Find Transfer Time of Flight -----------------
        IF ( e2 .lt. 0.9999D0 ) THEN
            Sinv = ( DSQRT( 1.0D0-e2*e2 )*DSin(Nu2) ) /
     &             ( 1.0D0 + e2*DCos(Nu2) )
            Cosv = ( e2+DCos(Nu2) ) / ( 1.0D0 + e2*DCos(Nu2) )
            E    = DATAN2( Sinv,Cosv )
            TOF  = DSQRT( A2**3 ) * ( E - e2*DSin(E) )
          ELSE
            IF ( DABS( e2-1.0D0 ) .lt. 0.000001D0 ) THEN
* Parabolic TOF
              ELSE
* Hyperbolic TOF
              ENDIF
          ENDIF

*         IF Show = 'Y' THEN
*           BEGIN
*             GoToXY( 5,23 )
*             Write( 'e2 ',e2:10:6,' E = ',E*rad:10:6,' a2 ',a2:10:6 )
*             Gotoxy( 5,24 )
*             Write( 'V1  ',v1:10:5,' V2a = ',v2a:10:6 )
*             Gotoxy( 5,25 )
*             Write( 'V2b ',v2b:10:5,' V3 = ',v3:10:6,' TOTAL ',
*                    (DelVa+DelVb):10:6 )
*             Gotoxy( 5,26 )
*             Write( 'Phi2 ',Phi2b*Rad:10:6,' Phi3 = ',Phi3*Rad:10:6 )
*           END

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                       SUBROUTINE GENERALCOPLANAR
*
*  This SUBROUTINE calculates the delta v's for a general coplanar transfer for
*    either circle to circle, or ellipse to ellipse.  The notation used is from
*    the initial orbit (1) at point a, transfer is made to the transfer orbit (2),
*    and to the final orbit (3) at point b.
*
*  Algorithm     :
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  19 Dec 1989
*
*  Inputs        :
*    R1          - Initial position magnitude           DU
*    R3          - Final position magnitude             DU
*    e1          - Eccentricity of first orbit
*    e3          - Eccentricity of final orbit
*    Nu1         - True Anomaly of first orbit          rad
*    Nu3         - True Anomaly of final orbit          rad
*
*  Outputs       :
*    DelVa       - Change in velocity at point a          DU / TU
*    DelVb       - Change in velocity at point b          DU / TU
*    TOF         - Time of Flight for the transfer      TU
*
*  Locals        :
*    SME1        - Specfic Mechanical Energy of first orbit    DU2 / TU
*    SME2        - Specfic Mechanical Energy of transfer orbit DU2 / TU
*    SME3        - Specfic Mechanical Energy of final orbit    DU2 / TU
*    V1          - Velocity of 1st orbit at point a            DU / TU
*    V2a         - Velocity of transfer orbit at point a       DU / TU
*    V2b         - Velocity of transfer orbit at point b       DU / TU
*    V3          - Velocity of final orbit at point b          DU / TU
*    a1          - Semi Major Axis of first orbit              DU
*    a2          - Semi Major Axis of Transfer orbit           DU
*    a3          - Semi Major Axis of final orbit              DU
*    E           - Eccentric anomaly of transfer orbit at b    rad
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    BMW          pg.
*
* ------------------------------------------------------------------------
*
```

```
      SUBROUTINE GeneralCoplanar ( R1,R3,e1,e2,e3,Nu1,Nu2a,Nu2b,Nu3,
     &                              DelVa,DelVb,TOF )
        IMPLICIT NONE

        REAL*8 R1,R3,e1,e2,e3,Nu1,Nu2a,Nu2b,Nu3,DelVa,
     &         DelVb,TOF

* ------------------------------ Locals ------------------------------
        REAL*8 SME1,SME2,SME3, V1,V2a,V2b,V3, a1,a2,a3,Phi1,Phi2a,
     &         Phi2b,Phi3, E,Eo,Sinv,Cosv

* ------------------------- Initialize values -----------------------
        a1  = (r1*(1.0D0+e1*DCos(Nu1))) / (1.0D0 - e1*e1 )
        IF ( DABS( e2-1.0D0 ) .gt. 0.000001D0 ) THEN
            a2   = (r1*(1.0D0+e2*DCos(Nu2a))) / (1.0D0 - e2*e2 )
            SME2 = -1.0D0 / (2.0D0*a2)
          ELSE
            a2   = 999999.9D0
* Undefined for Parabolic orbit
            SME2 = 0.0D0
          ENDIF
        a3  = (r3*(1.0D0+e3*DCos(Nu3))) / (1.0D0 - e3*e3 )
        SME1 = -1.0D0 / (2.0D0*a1)
        SME3 = -1.0D0 / (2.0D0*a3)

* ---------------------- Find Delta v at point a ---------------------
        V1  = DSQRT( 2.0D0*( (1.0D0/R1) + SME1 ) )
        V2a = DSQRT( 2.0D0*( (1.0D0/R1) + SME2 ) )
        Phi2a = DATAN( ( e2*DSin(Nu2a) ) / ( 1.0D0 + e2*DCos(Nu2a) ) )
        Phi1  = DATAN( ( e1*DSin(Nu1) ) / ( 1.0D0 + e1*DCos(Nu1) ) )
        DelVa= DSQRT( V2a*V2a + V1*V1 -
     &                2.0D0*V2a*V1*DCos( Phi2a-Phi1 ) )


* ---------------------- Find Delta v at point b ---------------------
        V3  = DSQRT( 2.0D0*( (1.0D0/R3) + SME3 ) )
        V2b = DSQRT( 2.0D0*( (1.0D0/R3) + SME2 ) )
        Phi2b = DATAN( ( e2*DSin(Nu2b) ) / ( 1.0D0 + e2*DCos(Nu2b) ) )
        Phi3  = DATAN( ( e3*DSin(Nu3) ) / ( 1.0D0 + e3*DCos(Nu3) ) )
        DelVb= DSQRT( V2b*V2b + V3*V3 -
     &                2.0D0*V2b*V3*DCos( Phi2b-Phi3 ) )

* -------------------- Find Transfer Time of Flight -----------------
        Sinv= ( DSQRT( 1.0D0-e2*e2 )*DSin(Nu2b) ) /
     &              ( 1.0D0 + e2*DCos(Nu2b) )
        Cosv= ( e2+DCos(Nu2b) ) / ( 1.0 + e2*DCos(Nu2b) )
        E= DATAN2( Sinv,Cosv )
        Sinv= ( DSQRT( 1.0D0-e2*e2 )*DSin(Nu2a) ) /
     &              ( 1.0D0 + e2*DCos(Nu2a) )
        Cosv= ( e2+DCos(Nu2a) ) / ( 1.0D0 + e2*DCos(Nu2a) )
        Eo= DATAN2( Sinv,Cosv )
        TOF= DSQRT( A2**3 )*( (E - e2*DSin(E)) - (Eo - e2*DSin(Eo)) )

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                          SUBROUTINE RENDEZVOUS
*
*  This subroutine calculates parameters for a hohmann transfer rendezvous.
*
*  Algorithm     : Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  23 Sep 1988
*
*  Inputs        :
*    Rcs1        - Radius of circular orbit interceptor        DU
*    Rcs2        - Radius of circular orbit target             DU
*    PhaseI      - Initial phase angle                         rad
*    NumRevs     - Number of revs to wait
*
*  Outputs       :
*    PhaseF      - Final Phase Angle                           rad
*    WaitTime    - Wait befor next intercept opportunity       TU
*
*  Locals        :
*    TOFTrans    - Time of flight of transfer orbit            TU
*    ATrans      - Semi-major axis of transfer orbit           DU
*    VelTgt      - Velocity of target                          rad / TU
*    VelInt      - Velocity of interceptor                     rad / TU
*    LeadAng     - Lead Angle                                  rad
*
*  Constants     :
*    Pi                                                         3.14159265358979
*
*  Coupling      :
*    None.
*
*  References    :
*    BMW           pg.
*
* ------------------------------------------------------------------------

      SUBROUTINE Rendezvous ( Rcs1,Rcs2,PhaseI,NumRevs,PhaseF,WaitTime)
        IMPLICIT NONE
        REAL*8 Rcs1,Rcs2,PhaseI,PhaseF,WaitTime
        INTEGER NumRevs

* ----------------------------- Locals ------------------------------
        REAL*8 TOFTrans,LeadAng,aTrans,VelTgt,VelInt,Pi

* ------------------------ Initialize values   ----------------------
        Pi     =  3.14159265358979D0

        ATrans  = (Rcs1 + RCS2) / 2.0D0
        TOFTrans= Pi*DSQRT( ATrans**3 )
        VelInt  = 1.0D0 / ( DSQRT(Rcs1**3) )
        VelTgt  = 1.0D0 / ( DSQRT(Rcs2**3) )

        LeadAng = VelTgt * TOFTRans
        PhaseF  = Pi - LeadAng
        WaitTime= ( PhaseI - PhaseF + 2.0D0*Pi*NumRevs ) /
     &            ( VelInt - VelTgt )
*
*        Write(*,*) '    A transfer  = ',ATRans, ' DU '
*        Write(*,*) '    TOF Transfer= ',TOFTRans,' TU '
*        Write(*,*) '    VelTgt      = ',VelTgt,  ' rad/TU'
*        Write(*,*) '    VelInt      = ',VelInt,  ' rad/TU'
*        Write(*,*) '    Lead Angle  = ',LeadAng*57.295779,' deg'
*
      RETURN
      END
*
```

```
* ----------------------------------------------------------------------------
*
*                         SUBROUTINE INTERPLANETARY
*
*  This subroutine calculates the delta v's for an interplanetary mission.  The
*    transfer assumes circular orbits for each of the planets.  Notice the
*    units are all metric since this procedure is designed for ANY planet and
*    sun system.  This eliminates having knowledge of cannonical units for
*    each planet in the calculations.
*
*  Algorithm     : Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  22 Mar 1989
*
*  Inputs        :
*    R1          - Radius of planet 1 from sun                  km
*    R2          - Radius of planet 2 from sun                  km
*    Rbo         - Radius at burnout about planet 1             km
*    Rimpact     - Radius at impact on planet 2                 km
*    Mu1         - Gravitational parameter of planet 1          km3/s2
*    Mut         - Gravitational parameter of planet Sun        km3/s2
*    Mu2         - Gravitational parameter of planet 2          km3/s2
*
*  Outputs       :
*    DeltV1      - Hyperbolic Excess velocity at planet 1 SOI    km/s
*    DeltV2      - Hyperbolic Excess velocity at planet 2 SOI    km/s
*    Vbo         - Burnout velocity at planet 1                 km/s
*    Vretro      - Retro velocity at surface of planet 2        km/s
*
*  Locals        :
*    SME1        - Specfic Mechanical Energy of 1st orbit       Km2/s
*    SMEt        - Specfic Mechanical Energy of transfer orbit  Km2/s
*    SME2        - Specfic Mechanical Energy of 2nd orbit       Km2/s
*    Vcs1        - Velocity of 1st orbit at delta v 1 point     km/s
*    Vcs2        - Velocity of 2nd orbit at delta v 2 point     Km/s
*    Vt1         - Velocity of Transfer orbit at delta v 1 point Km/s
*    Vt2         - Velocity of Transfer orbit at delta v 2 point Km/s
*    A           - Semi Major Axis of Transfer orbit            Km
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    BMW           pg.
*
* ----------------------------------------------------------------------------
*
```

```
      SUBROUTINE Interplanetary ( R1,R2,Rbo,Rimpact,Mu1,Mut,Mu2,
     &                            Deltv1,Deltv2,Vbo,Vretro    )
        IMPLICIT NONE
        REAL*8  R1,R2,Rbo,Rimpact,Mu1,Mut,Mu2,Deltv1,Deltv2,Vbo,
     &          Vretro

* ----------------------------- Locals ------------------------------
        REAL*8 SME1,SME2,SMEt, Vcs1, Vcs2, Vt1, Vt2, A

* --- Find a, SME, apogee and perigee velocities of transfer orbit ----
        A= (R1+R2) / 2.0D0
        SMEt= -Mut/ (2.0D0*A)
        Vt1= DSQRT( 2.0D0*( (Mut/R1) + SMEt ) )
        Vt2= DSQRT( 2.0D0*( (Mut/R2) + SMEt ) )

* ------- Find circular velocities of launch and target planets -------
        Vcs1= DSQRT( Mut/R1 )
        Vcs2= DSQRT( Mut/R2 )

* -------- Find delta velocities for Hohmann transfer protion ---------
        DeltV1= DABS( Vt1 - Vcs1 )
        DeltV2= DABS( Vcs2 - Vt2 )

* ----- Find SME and burnout/impact vel of launch/target planets ------
        SME1= Deltv1*DeltV1 / 2.0D0
        SME2= Deltv2*DeltV2 / 2.0D0
        Vbo    = DSQRT( 2.0D0*( (Mu1/Rbo) + SME1 ) )
        Vretro= DSQRT( 2.0D0*( (Mu2/Rimpact) + SME2 ) )

*         TP= Pi*DSQRT( a**3/Mut )
*         Write(*,*)  'Transfer Period = ',TP/3.1536E07,' yrs  or  ',
*     &               TP/86400.0,' days'
*         Write(*,*)  'Vcs  km/s',vcs1,' ',vcs2
*         Write(*,*)  ' Vt  km/s',vt1,' ',vt2
*         Write(*,*)  'SME km2/s2',SME1,' ',SMEt,' ',SME2

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                              SUBROUTINE REENTRY
*
*  This SUBROUTINE calculates various reentry parameters using the
*    Allen & Eggars approximations.
*
*  Algorithm     : Calculate the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  19 Dec 1989
*
*  Inputs        :
*    Vre         - Reentry Velocity                              m/s
*    PhiRe       - Reentry Flight Path Angle                     rad
*    BC          - Ballistic Coefficient                         kg/m2
*    h           - Altitude                                      km
*
*  Outputs       :
*    V           - Velocity                                      km/s
*    Decl        - Decleration                                   g's
*    MaxDecl     - Maximum Decleration                           g's
*
*  Locals        :
*    grav        - Temporary variable to hold Weight component
*    Rho         - Atmospheric density                           kg/m3
*
*  Constants        :
*    ScaleHt     - Scale height used to exponentially model atmosp 1.0/7.315
*
*  Coupling      :
*    None.
*
*  References    :
*    None.
*
* ----------------------------------------------------------------------------

      SUBROUTINE Reentry ( VRe,PhiRe,BC,H,  V,Decl,MaxDecl )
        IMPLICIT NONE

        REAL*8 VRe,PhiRe,BC,H, V,Decl,MaxDecl

* ----------------------------  Locals  -----------------------------
        REAL*8 ScaleHt,grav,Rho

* ------------------------  Implementation   ----------------------
        ScaleHt= 1.0D0/7.315D0

        Rho = 1.225D0*EXP( -ScaleHt*h )
        V   = Vre * EXP( (1000.0D0*Rho ) /
     &        (2.0D0*BC*ScaleHt*DSin(PhiRe)) )
        grav= 9.80D0*DSin(PhiRe)
        Decl= ((-0.5D0*Rho*V*V) / BC ) + grav
        Decl= Decl/9.80D0

        MaxDecl= (-0.5D0*ScaleHt*Vre*Vre*DSin(Phire)) /
     &           (9.80D0*EXP(1.0D0))
        MaxDecl= MaxDecl/9.80D0

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------------
*
*                           SUBROUTINE HILLSR
*
*  This SUBROUTINE calculates various position information for Hills equations.
*    Notice the XYZ system used has Y Colinear with Target Position vector,
*    Z normal to target orbit plane, and x in direction of velocity.
*
*  Algorithm     : Find the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  19 Dec 1989
*
*  Inputs        :
*    R           - Initial Position vector of INT        DU
*    V           - Initial Velocity Vector of INT        DU / TU
*    Alt         - Altitude of TGT satellite             DU
*    T           - Desired Time                          TU
*
*  Outputs       :
*    R1          - Final Position vector of INT          DU
*    V1          - Final Velocity Vector of INT          DU / TU
*
*  Locals        :
*
*  Constants     :
*
*  Coupling      :
*    None.
*
*  References    :
*    Kaplan        pg 108 - 115
*
* ----------------------------------------------------------------------------

      SUBROUTINE HillsR ( R,V,alt,t,  R1,V1 )
        IMPLICIT NONE

        REAL*8 R(4),V(4),alt,t,  R1(4),V1(4)

* ----------------------------- Locals ------------------------------
        REAL*8 SinNt,CosNt,Omega,nt,Radius

* -------------------- Initialize the orbit elemnts -------------------
        Radius= 1.0D0 + Alt
        Omega = DSQRT( 1.0D0 / Radius )
        nt    = Omega*t
        CosNt = DCos( nt )
        SinNt = DSin( nt )

* -------------------- Determine new positions  -----------------------
        R1(1)= ( 2.0D0*V(2)/Omega ) * CosNt +
     &           ( (4.0D0*V(1)/Omega) + 6.0D0*R(2) ) * SinNt +
     &           ( R(1) - (2.0D0*V(2)/Omega) ) -
     &           ( 3.0D0*V(1) + 6.0D0*Omega*R(2) )*t
        R1(2)= ( V(2)/Omega ) * SinNt -
     &           ( (2.0D0*V(1)/Omega) + 3.0D0*R(2) ) * CosNt +
     &           ( (2.0D0*V(1)/Omega) + 4.0D0*R(2) )
        R1(3)= R(3)*CosNt + (V(3)/Omega)*SinNt

* -------------------- Determine new velocities  ----------------------
        V1(2)= 0.0D0
        V1(1)= 0.0D0
        V1(3)= 0.0D0

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE HILLSV
*
*  This SUBROUTINE calculates initial velocity for Hills equations.
*    Notice the XYZ system used has Y Colinear with Target Position vector,
*    Z normal to target orbit plane, and x in direction of velocity.
*
*  Algorithm     : Check for a divide by zero, then
*                  Find the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  19 Dec 1989
*
*  Inputs        :
*    R           - Initial Position vector of INT        DU
*    Alt         - Altitude of TGT satellite             DU
*    T           - Desired Time                          TU
*
*  Outputs       :
*    V           - Initial Velocity Vector of INT        DU / TU
*
*  Locals        :
*
*  Constants     :
*
*  Coupling      :
*    None.
*
*  References    :
*    Kaplan        pg 108 - 115
*
* ---------------------------------------------------------------------------

      SUBROUTINE HillsV ( R,alt,t,  V )
        IMPLICIT NONE

        REAL*8 R(4),alt,t,  V(4)

* ----------------------------  Locals  -----------------------------
        REAL*8 Numer,Denom,SinNt,CosNt,Omega,nt,Radius

* -------------------- Initialize the orbit elemnts -------------------
        Radius= 1.0D0 + Alt
        Omega = DSQRT( 1.0D0 / Radius )
        nt    = Omega*t
        CosNt = DCos( nt )
        SinNt = DSin( nt )

* -------------------- Determine initial Velocity --------------------
        Numer= ( (6.0D0*r(2)*(nt-SINnT)-r(1))*Omega*Sinnt-2.0D0*Omega*
     &         r(2)*(4.0D0-3.0D0*Cosnt)*(1.0D0-COSnt) )
        Denom= (4.0D0*Sinnt-3.0D0*nt)*Sinnt +
     &         4.0D0*( 1.0D0-CosNt ) * ( 1.0D0-CosNt )

        IF ( DABS( Denom ) .gt. 0.000001D0 ) THEN
            V(1)= Numer / Denom
          ELSE
            V(1)= 0.0D0
          ENDIF
        IF ( DABS( SinNt ) .gt. 0.000001D0 ) THEN
            V(2)= -( Omega*r(2)*(4.0D0-3.0D0*Cosnt)+
     &                2.0D0*(1.0D0-Cosnt)*v(1) ) / ( SinNt )
          ELSE
            V(2)= 0.0D0
          ENDIF
        V(3)= 0.0D0

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE TARGET
*
*  This subroutine accomplishes the targeting problem using PKEPLER and GAUSS.
*    This routine uses J2 secular perturbations for moving the traget orbit.
*
*  Algorithm     : Propogate the target forward
*                  Find the intercept trajectory
*                  Calculate the change in velocity required
*
*  Author  : Capt Dave Vallado  USAFA/DFAS  719-472-4109   11 Sep 1990
*
*  Inputs  :
*    RInt        - Initial Position vector of Interceptor     DU
*    VInt        - Initial Velocity vector of Interceptor     DU/TU
*    RTgt        - Initial Position vector of Target          DU
*    VTgt        - Initial Velocity vector of Target          DU/TU
*    dm          - Direction of Motion for Gauss              'L','S'
*    TOF         - Time of flight to the intercept            TU
*
*  Outputs :
*    V1t         - Initial Transfer Velocity vector           DU/TU
*    V2t         - Final Transfer Velocity vector             DU/TU
*    DV1         - Initial Change Velocity vector             DU/TU
*    DV2         - Final Change Velocity vector               DU/TU
*
*  Local Variables  :
*    R1Tgt       - Position vector after TOF of Target        DU
*    V1Tgt       - Velocity vector after TOF of Target        DU/TU
*
*  Constants        :
*    None
*
*  Other Procedures :
*    PKEPLER       Find R and V at future time using J2 secular effects
*    GAUSS         Find velocity vectors at each end of transfer
*    LNCOM2        Linear combination of two vectors and constants
*
*  References       :
*    None.
*
* --------------------------------------------------------------------------- }

      SUBROUTINE TARGET( RInt,VInt,RTgt,VTgt,Dm,TOF, V1t,V2t,DV1,DV2 )
        IMPLICIT NONE
        REAL*8 RInt(4),VInt(4),RTgt(4),VTgt(4),TOF,V1t(4),V2t(4),
     +         DV1(4),DV2(4)
        CHARACTER*1 dm

* ------------------------- Local Variables -------------------------
        REAL*8 R1Tgt(4), V1Tgt(4)

* ---------------- Propogate Target forward by TOF -------------------
        CALL PKEPLER( RTgt,VTgt,TOF,  R1Tgt,V1Tgt )

* ---------------- Calculate transfer orbit between r's ---------------
        CALL GAUSS( RInt,R1Tgt,dm,TOF,  V1t,V2t )

        CALL LNCOM2( -1.0D0,1.0D0,VInt, V1t,  DV1 )
        CALL LNCOM2( 1.0D0,-1.0D0,V1Tgt,V2t,  DV2 )

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                          SUBROUTINE PKEPLER
*
*  This subroutine propagates a satellite's position and velocity vector over
*    a given time period accounting for perturbations caused by J2.  The
*    satellite's original position and velocity vectors are inputed together
*    with the time the elements are to be propagated for.  The updated position
*    and velocity vectors are then output.
*
*  Algorithm     : Find the value of the perturbations
*                  Determine the type of orbit
*                  Update the appropriate parameters
*                  Find the new position and velocity vectors
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   6 Jan 1990
*
*  Inputs        :
*    R           - original position vector                 DU
*    V           - original velocity vector                 DU/TU
*    DeltaT      - time for which orbital elements are to   TU
*
*  Outputs       :
*    R1          - updated position vector                  DU
*    V1          - updated velocity vector                  DU/TU
*
*  Locals        :
*    P           - Semi-paramter                            DU
*    A           - semimajor axis                           DU
*    E           - eccentricity
*    Inc         - inclination                              rad
*    Argp        - argument of periapsis                    rad
*    ArgpDot     - change in argument of periapsis          rad/TU
*    Omega       - longitude of the ascending node          rad
*    OmegaDot    - change in Omega                          rad
*    E0          - eccentric anomaly                        rad
*    E1          - eccentric anomaly                        rad
*    M           - mean anomaly                             rad/TU
*    MDot        - change in mean anomaly                   rad/TU
*    Uo          - argument of latitude                     rad
*    UDot        - change in argument of latitude           rad/TU
*    Lo          - true longitude of vehicle                rad
*    LDot        - change in the true longitude             rad/TU
*    CapPio      - longitude of periapsis                   rad
*    CapPioDot   - longitude of periapsis change            rad/TU
*    N           - mean angular motion                      rad/TU
*    NUo         - true anomaly                             rad
*    J2oP2       - J2 over p sqyared
*    Sinv,Cosv   - Sine and Cosine of Nu
*    Small       - Tolerance
*
*  Constants       :
*    J2          - J2 constant from the Earth's geopotential function
*    TwoPi       -
*    Pi          -
*
*  Coupling      :
*    ELORB       - Orbit Elements from position and Velocity vectors
*    RANDV       - Position and Velocity Vectors from orbit elements
*    NewtonR     - Newton Rhapson to find Nu and Eccentric anomaly
*
*  References    :
*    Escobal       pg 369. Dot terms
*    BMW           pg
*
* ---------------------------------------------------------------------------
*
```

```
      SUBROUTINE PKepler       ( R,V,DeltaT,      R1,V1     )
        IMPLICIT NONE
        REAL*8 R(4), V(4), DeltaT, R1(4), V1(4)
* -------------------------------- Locals --------------------------------
        REAL*8 P,A,E,Inc,Omega,Argp,Nuo,M,Uo,Lo,CapPio,OmegaDot,E0,
     &         ArgpDot,MDot,UDot,LDot,CapPiDot,N,J2oP2,TwoPi,
     &         Small,J2,NBar,Pi
        Character*5 TypeOrbit
* ------------------------- Implementation -------------------------
        TwoPi =  6.28318530717959D0
        Pi    =  3.14159265358979D0
        J2    =  0.00108263D0
        Small =  0.000001D0

        CALL ELORB( R,V,P,A,E,Inc,Omega,Argp,Nuo,M,Uo,Lo,CapPio)
        n = DSQRT(1.0D0/A**3)

* ----------------- Find the value of J2 perturbations ---------------
        J2oP2 = (1.5D0*J2) / (p**2)
        NBar= n*( 1.0D0 + J2oP2*DSQRT(1.0D0-e*e)*
     &                  (1.0D0 - 1.5D0*DSin(Inc)**2) )
        OmegaDot = -J2oP2 * DCos(inc) * NBar
        ArgpDot  =  J2oP2 * (2.0D0-2.5D0*DSin(inc)**2) * Nbar
        MDot     =  NBar
*       EDot     = -(4.0D0/3.0D0) * (1.0D0-E) * (MDot/Nbar)  Drag Terms

* -------------- Determine type of orbit for later use --------------
        TypeOrbit= 'EI'
        IF ( E.LT.Small ) THEN
* ----------------------- Circular Equatorial -----------------------
            IF (( Inc.LT.Small ).or.( DABS(Inc-Pi).LT.Small )) THEN
                TypeOrbit= 'CE'
              ELSE
* ----------------------- Circular Inclined ---------------------------
                TypeOrbit= 'CI'
              ENDIF
          ELSE
* ---------- Elliptical, Parabolic, Hyperbolic Equatorial ------------
            IF (( Inc.LT.Small ).or.( ABS(Inc-Pi).LT.Small )) THEN
                TypeOrbit= 'EE'
              ENDIF
          ENDIF

* -------- Update the orbital elements for each orbit type ------------
* ---------------------- Elliptical - Inclined ------------------------
        IF ( TypeOrbit.eq.'EI' ) THEN
              Omega  = Omega + OmegaDot * DeltaT
              Omega  = DMOD(Omega, TwoPi)
              Argp   = Argp  + ArgpDot  * DeltaT
              Argp   = DMOD(Argp, TwoPi)
              M      = M      + MDOT     * DeltaT
              M      = DMOD(M, TwoPi)
              CALL NewtonR( e,m,  e0,Nuo )
           ENDIF
* ---------------------- Circular - Inclined --------------------------
        IF ( TypeOrbit.eq.'CI' ) THEN
              Omega  = Omega + OmegaDot * DeltaT
              Omega  = DMOD(Omega, TwoPi)
              UDot   = ArgpDot + MDot
              Uo     = Uo     + UDot     * DeltaT
              Uo     = DMOD(Uo, TwoPi)
           ENDIF
* ---------------------- Elliptical - Equatorial ----------------------
        IF ( TypeOrbit.eq.'EE' ) THEN
              CapPiDot = OmegaDot + ArgpDot
              CapPio   = CapPio   + CapPiDot * DeltaT
              CapPio   = DMOD(CapPio, TwoPi)
              M       = M      + MDOT      * DeltaT
              M       = DMOD(M, TwoPi)
              CALL NewtonR( e,m,  e0,Nuo )
           ENDIF
* ---------------------- Circular - Equatorial ------------------------
        IF ( TypeOrbit.eq.'CE' ) THEN
              LDot = OmegaDot + ArgpDot + MDot
              Lo   = Lo      + LDot     * DeltaT
              Lo   = DMOD(Lo, TwoPi)
           ENDIF

* ----------------- Use RANDV to find new r and v vectors -----------
        CALL RANDV( P,E,Inc,Omega,Argp,Nuo,Uo,Lo,CapPio,  R1,V1 )
      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                            SUBROUTINE J2DRAGPERT
*
*  This subroutine calculates the perturbations for the predict problem
*    involving secular rates of change resulting from J2 and Drag only.
*
*  Algorithm     : Find the startup values
*                  Calculate the dot terms
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  28 Jan 1991
*
*  Inputs        :
*    Inc         - Inclination                          rad
*    e           - Eccentricity
*    N           - Mean Motion                          rad/TU
*    NDot        - Mean Motion rate                     rad / 2TU2
*
*  Outputs       :
*    OmegaDot    - Long of Asc Node rate                rad / TU
*    ArgpDot     - Argument of perigee rate             rad / TU
*    EDot        - Eccentricity rate                    / TU
*
*  Locals        :
*    P           - Semi-parameter                       DU
*    A           - Semi-major axis                      DU
*
*  Constants     :
*    J2            J2 zonal harmonic
*
*  Coupling      :
*    None.
*
*  References    :
*    Escobal       pg. 369
*    O'Keefe et al., Astronomical J, Vol 64 num 7, pg. 247 for Edot
*
* -----------------------------------------------------------------------------

      SUBROUTINE J2DragPert ( Inc,E,N,NDot,  OmegaDOT,ArgpDOT,EDOT )
        IMPLICIT NONE
        REAL*8 Inc,E,N,NDot, OmegaDOT,ArgpDOT,EDOT

* ----------------------------  Locals  ------------------------------
        REAL*8 P,A,J2,NBar

* ------------------------  Implementation   -------------------------
        J2  =  0.00108228D0

        a   = (1.0D0/n) ** (2.0D0/3.0D0)
        p   = a*(1.0D0 - e**2)
        NBar= n*( 1.0D0+1.5D0*J2*(DSQRT(1.0D0-e*e)/(p*p))*
     &           ( 1.0D0-1.5D0*DSin(inc)**2 ))

* ----------------------- Find dot Terms ---------------------------
        OmegaDot = -1.5D0*( J2/(p*p) ) * DCos(inc) * NBar
        ArgpDot  =  1.5D0*( J2/(p*p) ) * (2.0D0-2.5D0*DSin(inc)**2) *
     &                 Nbar
        EDot     = -(4.0D0/3.0D0) * (1.0D0-E) * (NDot/Nbar)

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                          SUBROUTINE PREDICT
*
*  This subroutine determines the azimuth and elevation for the viewing
*    of a staellite from a known ground site.  Notice the Julian Date is left
*    in it's usual DAYS format since the dot terms are input as radians per
*    day, thus no extra need for conversion.  The Julian Date also facilitates
*    finding the site position vector.  Also observe RANDV is not used since
*    this would merely accomplish extra calculations.  The iteration is left
*    out to allow the user to set up his own loop to look for sighting times.
*
*  Algorithm     :
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs        :
*    JD          - Julian Date of desired observation      Days
*    JDEpoch     - Julian date of epoch for satellite      Days
*    No          - Epoch Mean motion                       rad/day
*    Ndot        - Epcoh Half Mean Motion Rate             rad/2day2
*    Eo          - Epoch Eccentricity
*    Edot        - Epoch Eccentricity rate                 /day
*    Inco        - Epoch Inclination                       rad
*    Omegao      - Epoch Lon of Asc node                   rad
*    OmegaDot    - Epoch Lon of Asc Node rate              rad/day
*    Argpo       - Epoch Argument of perigee               rad
*    ArgpDot     - Epoch Argument of perigee rate          rad/day
*    Mo          - Epoch Mean Anomaly                      rad
*    Lat         - Geodetic Latitude of site               rad
*    Lon         - Longitude of site                       rad
*    Alt         - Altitude of site                        DU
*
*  Outputs       :
*    Rho         - Range from site to satellite            DU
*    Az          - Azimuth                                 rad
*    El          - Elevation                               rad
*    Vis         - Visibility Flag 'Radar Sun','Radar Nite','Eye','Not Visible'
*
*  Locals        :
*    Variable o  - denotes the epoch value, while no o is current
*    Dt          - Change in time from Epoch to desired t days
*    A           - Semi major axis                         DU
*    E0          - Eccentric Anomaly                       rad
*    Nu          - True Anomaly                            rad
*    LST         - Local Sidereal Time                     rad
*    GST         - Greenwich Sidereal Time                 rad
*    Temp        - Temporary Real value
*    RtAsc       - Suns Right ascension                    rad
*    Decl        - Suns Declination                        rad
*    Theta       - Angle between IJK Sun and Satellite vecrad
*    Dist        - Ppdculr distance of satellite from RSunDU
*    Small       - Tolerance of small values
*    R           - IJK Satellite vector                    DU
*    RS          - IJK Site Vector                         DU
*    VS          - Site Velocity vector                    DU/TU
*    RhoVec      - Site to satellite vector in SEZ         DU
*    TempVec     - Temporary vector
*    RHoV        - Site to satellite vector in IJK         DU
*    RSun        - Sun vector                              AU
*    C           - Temporary Vector
*
*  Constants     :
*    Pi          -                                         3.14159265358979
*    HalfPi      -                                         1.57079632679490
*    TwoPi       -                                         6.28318530717959
*    TUDay       - Days in one TU                         0.00933809102919444
*    AUDU        - DUs in 1 AU                            23455.07
*
*  Coupling      :
*    SUN           Position vector of Sun
*    MAG           Magnitude of a vector
*    DOT           Dot product of two vectors
*    CROSS         Cross Product of two vectors
*    ROT1,ROT2,ROT3 Rotations about 1st, 2nd and 3rd axis
*    SITE          Site Vector
*    LSTime        Local Sidereal Time
*    NewtonR       Iterate to find Eccentric Anomaly
*    LNCOM1        Linear combination of a scalar and vector
*  References    :
*    Escobal       pg. 369
*
* ------------------------------------------------------------------------
*
```

```
      SUBROUTINE Predict ( JD,JDEpoch,no,Ndot,Eo,Edot,inco,Omegao,
     &                     OmegaDot,Argpo,ArgpDot,Mo,Lat,Lon,Alt,
     &                     Rho,Az,El,RtAsc,Decl,Vis )
        IMPLICIT NONE
        REAL*8 JD,JDEpoch,no,Ndot,Eo,Edot,inco,Omegao,RtAsc,Decl,
     &         OmegaDot,Argpo,ArgpDot,Mo,Lat,Lon,Alt,Rho,Az,El
        CHARACTER*11 Vis
        EXTERNAL Dot
* ----------------------------- Locals -----------------------------
        REAL*8 Dt,a,E0,Nu,LST,GST,Temp,Theta,Dist,AUDU,
     &         Small,Pi,HalfPi,TwoPi,TUDay,N,M,E,Omega,Argp,
     &         R(4),Rpqw(4),RS(4),VS(4),RhoVec(4),TempVec(4),RhoV(4),
     &         RSun(4),C(4),Dot
        INTEGER i

* ------------------------- Implementation -------------------------
        Small   =  0.000001D0
        Pi      =  3.14159265358979D0
        HalfPi  =  1.57079632679490D0
        TwoPi   =  6.28318530717959D0
        TUDay   =  0.00933809102919444D0
        AUDU    = 23455.07003D0
        Az      =  0.0D0
        El      =  0.0D0
        Rho     =  0.0D0
* ------------------- Update elements to new time -------------------
        Dt     = JD - JDEpoch
        e      = eo      + EDot*Dt
        Omega= Omegao + OmegaDot*Dt
        Argp  = Argpo  + ArgpDot*Dt
        M      = Mo      + No*Dt     + NDot*Dt*Dt
        M      = DMOD( M,TwoPi )
        N      = No + 2.0D0*NDot*Dt
        N      = N * TUDay
* ---------------- Newton Rhapson to find True Anomaly ---------------
        CALL NewtonR( e,M, E0,Nu )
* ---------------------- Form PQW position vector --------------------
        a= ( 1.0D0/(N*N) ) ** ( 1.0D0/3.0D0 )

        Rpqw(4)= ( a*(1.0D0-e*e) ) / (1.0D0 + e*DCos( Nu ) )
        Rpqw(1)= Rpqw(4)*DCos( Nu )
        Rpqw(2)= Rpqw(4)*DSin( Nu )
        Rpqw(3)= 0.0D0

* -------------------------- Rotate to IJK --------------------------
        CALL ROT3( Rpqw    , -Argp , TempVec )
        CALL ROT1( TempVec, -Inco , TempVec )
        CALL ROT3( TempVec, -Omega,    R      )

        CALL LSTIME( Lon,JD,      Lst,Gst )
        CALL SITE( Lat,Alt,Lst,  RS,VS )

* ----------- Find IJK range vector from site to satellite ------------
        DO i=1,3
            RhoV(i) =  R(i) - RS(i)
        ENDDO
        CALL MAG( RhoV )
        Rho= RhoV(4)

* --------- Calculate Topocentric Rt ascension and declination --------
        Temp= DSQRT( RhoV(1)**2 + RhoV(2)**2 )
        IF ( DABS( RhoV(2) ).LT.Small ) THEN
            IF ( Temp.LT.Small ) THEN
                RtAsc= 999999.1D0
              ELSE
                IF ( RhoV(1).GT.0.0D0 ) THEN
                    RtAsc= Pi
                  ELSE
                    RtAsc= 0.0D0
                  ENDIF
              ENDIF
          ELSE
            RtAsc  = DATAN2( RhoV(2)/Temp, RhoV(1)/Temp )
          ENDIF

        IF ( Temp.LT.Small ) THEN
            Decl= HalfPi
          ELSE
            Decl= DASIN( RhoV(3)/RhoV(4) )
          ENDIF
* -------------------------- Rotate to SEZ --------------------------
        CALL ROT3( RhoV,       LST    ,  TempVec )
        CALL ROT2( TempVec,HalfPi-Lat,   RhoVec )
*
```

```
* -------------------- Check visibility constraints --------------------
* ----------------------- Is it above the horizon ---------------------
      IF ( RhoVec(3).GT.0.0D0) THEN

* -------------- Is the site in the light, or in the dark -------------
          CALL SUN( JD,RSun,RtAsc,Decl )
          CALL LnCOM1( AUDU,RSun,  RSun )
          IF ( DOT( RSun,RS ).GT.0.0D0 ) THEN
              Vis = 'Radar  Sun '
            ELSE

* -------------- Is the satellite in the shadow or not  ---------------
              CALL CROSS( RSun, R, C )
              Theta= [illegible]in( C(4)/ (RSun(4)*R(4)) )
              Dist = [illegible]*DCOS( Theta - HalfPi )
              IF ( Dist.GT.1.0D0 ) THEN
                  Vis =  'Eye        '
                ELSE
                  Vis = 'Radar  Nite'
                ENDIF
            ENDIF
        ELSE
          Vis = 'Not Visible'
        ENDIF

* ------------------ Calculate the azimuth and elevation --------------
      Temp= DSQRT( RhoVec(1)**2 + RhoVec(2)**2 )
      IF ( DABS( RhoVec(2) ).LT.Small ) THEN
          IF ( Temp.LT.Small ) THEN
              Az= 999999.1D0
            ELSE
              IF ( RhoVec(1).GT.0.0D0 ) THEN
                  Az= Pi
                ELSE
                  Az= 0.0D0
                ENDIF
            ENDIF
        ELSE
          Az  = DATAN2( RhoVec(2)/Temp,-RhoVec(1)/Temp )
          IF (Az.LT.0.0D0) THEN
              Az= Twopi+Az
            ENDIF
        ENDIF

      IF ( Temp.LT.Small ) THEN
          El = HalfPi
        ELSE
          El = DASIN( RhoVec(3)/Rho )
        ENDIF

* --------- Calculate Geocentric Rt ascension and declination ---------
*     Temp= DSQRT( R(1)**2 + R(2)**2 )
*     IF ( DABS( R(2) ).LT.Small ) THEN
*         IF ( Temp.LT.Small ) THEN
*             RtAsc= 999999.1D0
*           ELSE
*             IF ( R(1).GT.0.0D0 ) THEN
*                 RtAsc= Pi
*               ELSE
*                 RtAsc= 0.0D0
*               ENDIF
*           ENDIF
*       ELSE
*         RtAsc  = DATAN2( R(2)/Temp, R(1)/Temp )
*       ENDIF
*
*     IF ( Temp.LT.Small ) THEN
*         Decl= HalfPi
*       ELSE
*         Decl= DASIN( R(3)/R(4) )
*       ENDIF

    RETURN
    END
*
```

```
* -----------------------------------------------------------------------------
*
*                             SUBROUTINE DERIV
*
*  This subroutine calculates the derivative of the state vector for use with
*    the Runge-Kutta algorithm.  This models the two-body EOM.
*
*  Algorithm     : Find the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   6 Apr 1989
*
*  Inputs        :
*    Time        - Time                                 TU
*    X           - State Vector                         DU, DU/TU
*
*  Outputs       :
*    XDot        - Derivative of State Vector           DU/TU, DU/TU2
*
*  Locals        :
*    X4Cubed     - Cube of X(4)
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    None.
*
* -----------------------------------------------------------------------------

      SUBROUTINE Deriv ( Time,X,  XDot )
        IMPLICIT NONE
        REAL*8 Time, X(6), XDot(6)

* ------------------------------ Locals -------------------------------
        Real*8 X4Cubed

* -------------------------- Implementation --------------------------
        X4Cubed= ( DSQRT( X(1)**2 + X(2)**2 + X(3)**2 ) )**3

* ------------------------ Velocity Terms ----------------------------
        XDot(1)= X(4)
        XDot(2)= X(5)
        XDot(3)= X(6)

* ---------------------- Acceleration Terms --------------------------
        XDot(4)= -X(1) / X4Cubed
        XDot(5)= -X(2) / X4Cubed
        XDot(6)= -X(3) / X4Cubed

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                          SUBROUTINE PERTACCEL
*
*  This subroutine calculates the actual value of the perturbing acceleration.
*
*  Algorithm     : Setup temporary values
*                  Use a case statement to select which perturbations are added
*                  Note each perturbation adds on to the previous result
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs        :
*    R           - Radius vector                          DU
*    V           - Velocity vector                        DU/TU
*    Time        - Initial time                           TU
*    WhichOne    - Which perturbation to calculate        1 2 3 4 5 ...
*    BC          - Ballistic Coefficient                  kg/m2
*
*  Outputs       :
*    APert       - Perturbing acceleration                DU/TU2
*
*  Locals        :
*    rs2         - Sun radius vector **2
*    rs3         - Sun radius vector **3
*    rm2         - Moon radius vector **2
*    rm3         - Moon radius vector **3
*    r32         - "z" component of Radius vector **2
*    r33         - "z" component of Radius vector **3
*    r34         - "z" component of Radius vector **4
*    r2          - Radius vector **2
*    r3          - Radius vector **3
*    r4          - Radius vector **4
*    r5          - Radius vector **5
*    r7          - Radius vector **7
*    Beta        -
*    Temp        - Temporary Real Value
*    rho         - Atmospheric Density
*    Va          - Relative Velocity Vector               DU / TU
*    RSun        - Radius Vector to Sun                   AU
*    RMoon       - Radius Vector to Moon                  DU
*    RtAsc       - Right Ascension                        deg
*    Decl        - Declination                            deg
*    i           - Index
*
*  Constants       :
*    J2          -                                         0.00108263
*    J3          -                                        -0.00000254
*    J4          -                                        -0.00000161
*    GMS         - Sun Gravitational Parameter DU3/TU2     332952.9364
*    GMM         - Moon Gravitational Parameter DU3/TU2    0.01229997
*    OmegaEarth  - Angular rotation of Earth (Rad/TU)      0.0588335906868878
*    TUDay       - Days in one TU                          0.00933809102919444
*
*  Coupling      :
*    MAG           Magnitude of a vector
*    Sun           Sun vector
*    Moon          Moon vector
*
*  References    :
*    None.
*
* ---------------------------------------------------------------------------
*
```

```
      SUBROUTINE PertAccel ( R,V,Time,WhichOne,BC,  APert )
        IMPLICIT NONE

        REAL*8 R(4),V(4),Time,APert(4),BC
        INTEGER WhichOne

* ---------------------------- Locals ------------------------------
        REAL*8 J2,J3,J4,gms,gmm,OmegaEarth,RtAsc,Decl,
     &    rs2,rm2,rs3,rm3,r32,r33,r34,r2,r3,r4,r5,r7,TUDay,
     &    Beta,Temp,rho,Vs(4),RSun(4),RMoon(4)
        INTEGER i

* ------------------------ Implementation  -----------------------
        OmegaEarth= 0.0588335906868878 6D0
        TUDay     = 0.00933809102919444D0
        CALL MAG( R )
        CALL MAG( V )

        R2  = r(4)**2
        R3  = R2*r(4)
        R4  = R2*R2
        R5  = R2*R3
        R7  = R5*R2
        R32 = r(3)*r(3)
        R33 = R32*r(3)
        R34 = R32*R32

* ---------------------- J2 Acceleration  ------------------------
        IF ( WhichOne.Eq.1 ) THEN
            J2  =  0.00108263D0
            APert(1)= ( (-1.5*J2*r(1)) / R5 ) * ( 1.0-(5.0*R32) / R2 )
            APert(2)= ( (-1.5*J2*r(2)) / R5 ) * ( 1.0-(5.0*R32) / R2 )
            APert(3)= ( (-1.5*J2*r(3)) / R5 ) * ( 3.0-(5.0*R32) / R2 )
            CALL MAG( APert )
          ENDIF

* --------------------- J3 Acceleration  ------------------------
        IF ( WhichOne.Eq.2 ) THEN
            J3  = -0.00000254D0
            APert(1)= ( (-2.5*J3*r(1)) / R7 ) * ((3.0*r(3))-(7.0*R33)
     &                  / R2 )
            APert(2)= ( (-2.5*J3*r(2)) / R7 ) * ((3.0*r(3))-(7.0*R33)
     &                  / R2 )
            IF (DABS( r(3) ).gt. 0.0000001) THEN
                APert(3)= ( (-2.5*J3*r(3)) / R7 ) * ((6.0*r(3))-
     &                      ((3.0*R33) / R2) - ((3.0*r2)  / r(3)))
              ELSE
                APert(3)= 0.0D0
              ENDIF
            CALL MAG( APert )
          ENDIF

* --------------------- J4 Acceleration  -- ---------------------
        IF ( WhichOne.Eq.3 ) THEN
            J4  = -0.00000161D0
            APert(1)= ( (-1.875*J4*r(1)) / R7 )*(1.0-((14.0*R32)/R2)+
     &                  ((21.0*R34) / R4 ))
            APert(2)= ( (-1.875*J4*r(2)) / R7 )*(1.0-((14.0*R32)/R2)+
     &                  ((21.0*R34) / R4 ))
            APert(3)= ( (-1.875*J4*r(3)) / R7 )*(5.0-((70.0*R32)/
     &                (3.0*R2))+((21.0*R34) / R4 ))
            CALL MAG( APert )
          ENDIF
*
```

```
* ---------------------- Sun Acceleration ------------------------
        IF ( WhichOne.Eq.4 ) THEN
            GMS =  3.329529364D05
            Temp = Time*TUDay
            CALL SUN( Temp,  RSun,RtAsc,Decl )

            DO i= 1,4
                RSun(i)= RSun(i)*23455.07003D0
            ENDDO

            RS2= RSun(4)**2
            RS3= RS2*RSun(4)
            APert(1)= (-GMS/RS3) *
     &                (r(1)-3.0*RSun(1)*
     &               ((r(1)*RSun(1)+r(2)*RSun(2)+r(3)*RSun(3)) / RS2))
            APert(2)= (-GMS/RS3) *
     &                (r(2)-3.0*RSun(2)*
     &               ((r(1)*RSun(1)+r(2)*RSun(2)+r(3)*RSun(3)) / RS2))
            APert(3)= (-GMS/RS3) *
     &                (r(3)-3.0*RSun(3)*
     &               ((r(1)*RSun(1)+r(2)*RSun(2)+r(3)*RSun(3)) / RS2))
            CALL MAG( APert )
          ENDIF

* ---------------------- Moon Acceleration ------------------------
        IF ( WhichOne.Eq.5 ) THEN
            GMM =  0.01229997D0
            Temp = Time*TUDay
            CALL MOON( Temp,  RMoon,RtAsc,Decl )
            RM2= RMoon(4)**2
            RM3= RM2*RMoon(4)
            APert(1)= (-GMM/RM3) *
     &                (r(1)-3.0*RMoon(1)*
     &                ((r(1)*RMoon(1)+r(1)*RMoon(1)+r(3)*RMoon(3))
     &                / RM2))
            APert(2)= (-GMM/RM3) *
     &                (r(2)-3.0*RMoon(2)*
     &                ((r(1)*RMoon(1)+r(2)*RMoon(2)+r(3)*RMoon(3))
     &                / RM2))
            APert(3)= (-GMM/RM3) *
     &                (r(3)-3.0*RMoon(3)*
     &                ((r(1)*RMoon(1)+r(3)*RMoon(3)+r(3)*RMoon(3))
     &                / RM2))
            CALL MAG( APert )
          ENDIF

* ---------------------- Drag Acceleration ------------------------
        IF ( WhichOne.Eq.6 ) THEN
            Va(1)= V(1) + (OmegaEarth*r(2))
            Va(2)= V(2) - (OmegaEarth*r(1))
            Va(3)= V(3)
            CALL MAG( Va )

            CALL ATMOS( R, Rho )

            Temp= -1000.0D0 * Va(4) * 0.5D0*Rho* ( 1.0D0/BC )
     &              * 6378137.0D0
            APert(1)= Temp*Va(1)
            APert(2)= Temp*Va(2)
            APert(3)= Temp*Va(3)
            CALL MAG( APert )
          ENDIF

* ---------------------- Solar Acceleration ------------------------
        IF ( WhichOne.Eq.7 ) THEN
            Temp = Time*TUDay
            CALL SUN( Temp,  RSun,RtAsc,Decl )
            Beta = 0.4D0
            APert(4)= (4.74D-06*(1.0+Beta))/(BC*9.807)
            APert(1)= (-APert(4)*RSun(1))/RSun(4)
            APert(2)= (-APert(4)*RSun(2))/RSun(4)
            APert(3)= (-APert(4)*RSun(3))/RSun(4)
          ENDIF

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                            SUBROUTINE PDERIV
*
*  This subroutine calculates the derivative state vector for the RK4
*    subroutine.  The DerivType string is used to determine which perturbation
*    equations are used.
*
*  Algorithm     : Assign values
*                  Check each value of Derivtype and if a perturbation is needed
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs        :
*    Time        - Initial time                              TU
*    X           - Initial state vector                      DU , DU/TU
*    DerivType   - String for including YN for Perts         YNNYYYNNY
*    BC          - Ballistic Coefficient                     kg/m2
*
*  Outputs       :
*    XDot        - Derivative of X                           DU/TU , DU/TU2
*
*  Locals        :
*    RCubed      - Radius vector cubed                       DU3
*    Ro          - Radius vector                             DU
*    Vo          - Velocity vector                           DU/TU
*    APert       - Perturbing acceleration                   DU/TU2
*    TempPert    - Temporary acceleration                    DU/TU2
*    i           - Index
*
*  Constants     :
*    None.
*
*  Coupling      :
*    PertAccel     Calculates the actual values of each perturbing acceleration
*    AddVec        Adds two vectors together
*
*  References    :
*    None.
*
* -----------------------------------------------------------------------------
*
```

```
      SUBROUTINE PDERIV( Time,X,DerivType,BC,  XDot )
        IMPLICIT NONE
        REAL*8 X(6),XDot(6),Time,BC
        CHARACTER*10 DerivType

* ------------------------------ Locals ------------------------------
        REAL*8 RCubed,Ro(4),Vo(4),APert(4),TempPert(4)
        INTEGER i

* -------------------------- Implementation  --------------------------
        DO i= 1, 3
            APert(i)= 0.0D0
            Ro(i)   = X(i)
            Vo(i)   = X(i+3)
        ENDDO
        CALL MAG( Ro )
        CALL MAG( Vo )
        APert(4)= 0.0D0
        RCubed = Ro(4)**3

* -------------------------- Velocity Terms --------------------------
        XDot(1)= X(4)
        XDot(2)= X(5)
        XDot(3)= X(6)

* ------------------------ Acceleration Terms ------------------------
        IF ( DerivType(1:1).eq.'Y' ) THEN
            CALL PertAccel( Ro,Vo,Time,1,BC, APert )
          ENDIF
        IF ( DerivType(1:2).eq.'Y' ) THEN
            CALL PertAccel( Ro,Vo,Time,2,BC, TempPert )
            CALL AddVec( TempPert,APert,APert )
          ENDIF
        IF ( DerivType(1:3).eq.'Y' ) THEN
            CALL PertAccel( Ro,Vo,Time,3,BC, TempPert )
            CALL AddVec( TempPert,APert,APert )
          ENDIF
        IF ( DerivType(1:4).eq.'Y' ) THEN
            CALL PertAccel( Ro,Vo,Time,4,BC, TempPert )
            CALL AddVec( TempPert,APert,APert )
          ENDIF
        IF ( DerivType(1:5).eq.'Y' ) THEN
            CALL PertAccel( Ro,Vo,Time,5,BC, TempPert )
            CALL AddVec( TempPert,APert,APert )
          ENDIF
        IF ( DerivType(1:6).eq.'Y' ) THEN
            CALL PertAccel( Ro,Vo,Time,6,BC, TempPert )
            CALL AddVec( TempPert,APert,APert )
          ENDIF
        IF ( DerivType(1:7).eq.'Y' ) THEN
            CALL PertAccel( Ro,Vo,Time,7,BC, TempPert )
            CALL AddVec( TempPert,APert,APert )
          ENDIF

        XDot(4)= (-X(1) / RCubed) + APert(1)
        XDot(5)= (-X(2) / RCubed) + APert(2)
        XDot(6)= (-X(3) / RCubed) + APert(3)

      RETURN
      END
*
```

```
*  ---------------------------------------------------------------------------
*
*                          SUBROUTINE RK4
*
*  This subroutine is a fourth order Runge-Kutta integrator for an
*    N-dimensional First Order differential equation.  The user must provide
*    an external subroutine containing the system Equations of Motion.  Notice
*    time is included since some applications may need this.  The LAST position
*    in DerivType is a flag for two-body motion.  Two-Body motion is used if
*    the 10th element is set to '2', otherwise the Yes and No values determine
*    which perturbations to use.
*
*  Algorithm     : Evaluate each term depending on the derivtype
*                  Find the final answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    ITime       - Initial Time                          TU
*    DT          - Step size                             TU
*    N           - Dimension of the state
*    DerivType   - String for including YN for Perts     YNNYYYNNY2
*    BC          - Ballistic Coefficient                 kg/m2
*    X           - State vector at initial time          DU, DU/TU
*
*  Outputs       :
*    X           - State vector at new time              DU, DU/TU
*
*  Locals        :
*    XDot        - Derivative of State Vector            DU/TU,  DU/TU2
*    Time        - Time                                  TU
*    K           - Storage
*    TEMP        - Storage
*    J           - Index
*    TempTime    - Temporary time storage                TU
*
*  Constants     :
*    None.
*
*  Coupling      :
*    Deriv         Subroutine for Derivatives of E.O.M. for Two-Body problem
*    PDeriv        Subroutine for Perturbed E.O.M.
*
*  References    :
*    James, et al., "Applied Numerical Methods" pg. 461-466, eqtn pg. 463.
*    BMW            pg. 414-415
*
*  ---------------------------------------------------------------------------
*
```

```
      SUBROUTINE RK4 ( ITime,DT,N,DerivType,BC,  X )
        IMPLICIT NONE
        INTEGER N
        REAL*8  DT, ITime, X(N), BC
        CHARACTER*10 DerivType

* ------------------------------- Locals ------------------------------
        REAL*8 XDot(6), K(6,3), TEMP(6), Time, TempTime
        INTEGER J

* -------------------------- Implementation  ------------------------
          TempTime = ITime
          IF (DerivType(1:10).eq.'2') THEN
              CALL DERIV( ITime,X,XDot )
            ELSE
              CALL PDERIV( ITime,X,DerivType,BC,XDot )
            ENDIF

* -------------------- Evaluate 1st Taylor Series Term -----------------
          DO J = 1,N
             K(J,1)  = Dt * XDot(J)
             TEMP(J) = X(J) + 0.5D0*K(J,1)
          ENDDO

          Time = ITime + Dt/2.0D0

          IF (DerivType(1:10).eq.'2') THEN
              CALL DERIV( Time,Temp,XDot )
            ELSE
              CALL PDERIV( Time,Temp,DerivType,BC,XDot )
            ENDIF

* -------------------- Evaluate 2nd Taylor Series Term -----------------
          DO J = 1,N
             K(J,2)  = Dt * XDot(J)
             TEMP(J) = X(J) + 0.5D0*K(J,2)
          ENDDO

          IF (DerivType(1:10).eq.'2') THEN
              CALL DERIV( Time,Temp,XDot )
            ELSE
              CALL PDERIV( Time,Temp,DerivType,BC,XDot )
            ENDIF

* -------------------- Evaluate 3rd Taylor Series Term -----------------
          DO J = 1,N
             K(J,3)  = Dt * XDot(J)
             TEMP(J) = X(J) + K(J,3)
          ENDDO

          TempTime = TempTime + Dt

          IF (DerivType(1:10).eq.'2') THEN
              CALL DERIV( TempTime,Temp,XDot )
            ELSE
              CALL PDERIV( TempTime,Temp,DerivType,BC,XDot )
            ENDIF

* ---------------------- Update the state vector  ---------------------
          DO J = 1,N
             X(J) = X(J) + ( K(J,1) + 2.0D0*(K(J,2) + K(J,3)) +
     &                         Dt*XDot(J) ) / 6.0D0
          ENDDO

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                           SUBROUTINE ATMOS
*
*  This subroutine finds the atmospheric density at an altitude above an
*    oblate earth given the position vector in the Geocentric Equatorial
*    frame.  The position vector is in DU's and the density is in gm/cm**3.
*
*  Algorithm     : Find initial values
*                  Loop to find the latitudes
*                  Calculate the density through a cascading IF statement
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  20 Sep 1990
*
*  Inputs        :
*    R           - GEC Position vector                      DU
*
*  Outputs       :
*    Rho         - Density                                  gm/cm**3
*
*  Locals        :
*    Rc          - Range of site w.r.t. earth center        DU
*    Height      - Height above earth w.r.t. site           DU
*    Alt         - Altitude above earth w.r.t. site         km
*    OldDelta    - Previous value of DeltaLat               rad
*    DeltaLat    - Diff between Delta and Geocentric lat    rad
*    GeoDtLat    - Geodetic Latitude                       -Pi/2 to Pi/2 rad
*    GeoCnLat    - Geocentric Latitude                     -Pi/2 to Pi/2 rad
*    TwoFMinusF2 - 2*F - F squared
*    OneMinusF2  - ( 1 - F ) squared
*    Delta       - Declination angle of R in IJK system     rad
*    Temp        - Diff between Geocentric/Geodetic lat     rad
*    RSqrd       - Magnitude squared                        DU2
*    SinTemp     - Sine of Temp                             rad
*    RhoNom      - Nominal density at particular alt        gm/cm**3
*    H           - Scale Height                             km
*    i           - index
*
*  Constants     :
*    Pi          -                                          3.14159265358979
*    Flat        - Flatenning of the Earth                  0.003352810664747352
*    REarthKm    - Radius of Earth in km                    6378.137
*
*  Coupling      :
*    MAG           Magnitude of a vector
*
*  References    :
*    Escobal       pg. 398-399 ( Conversion to Lat and Height )
*    Blitzer       pg. 63 ( Atmospheric density )
*    Wertz         pg. 820 ( Low altitude density )
*
* -----------------------------------------------------------------------------
*
```

```
      SUBROUTINE ATMOS ( R,  Rho )
        IMPLICIT NONE
        REAL*8 R(4), Rho

* ----------------------------- Locals -----------------------------
        REAL*8 Rc, Height, OldDelta, DeltaLat, GeoDtLat,
     &         TwoFMinusF2,OneMinusF2, Delta, RSqrd, Temp, Pi, Flat,
     &         GeoCnLat,H, Rhonom, Alt, REarthkm, SinTemp
        INTEGER i

* ------------------------ Initialize values  ------------------------
        Pi     = 3.14159265358979D0
        CALL MAG( R )
        Flat   = 0.00335281066474 7352D0
        TwoFMinusF2 = 2.0D0*Flat - Flat**2
        OneMinusF2  = ( 1.0D0-Flat )**2
        REarthKm    = 6378.137D0

* ------------------- Set up initial latitude value -------------------
        Delta  = DATan( R(3) / DSQRT( R(1)*R(1) + R(2)*R(2) ) )
        IF ( DABS(Delta).GT.Pi ) THEN
            Delta = DMOD( Delta,Pi )
          ENDIF
        GeoCnLat = Delta
        OldDelta =  1.0D0
        DeltaLat = 10.0D0
        RSqrd    = R(4)**2

* -------- Iterate to find Geocentric and Geodetic Latitude  ----------
        i = 1
        DO WHILE ( ( DABS(OldDelta-DeltaLat).GT.0.00001D0).and.
     &         (i.LT.10) )
            OldDelta = DeltaLat
            Rc       = DSQRT( ( 1.0D0-TwoFMinusF2 ) /
     &                 ( 1.0D0-TwoFMinusF2*DCOS(GeoCnLat)**2 ) )
            GeoDtLat = DATan( DTAN(GeoCnLat) / OneMinusF2 )
            Temp     = GeoDtLat-GeoCnLat
            SinTemp  = DSIN( Temp )
            Height   = DSQRT( RSqrd - (Rc**2)*SinTemp**2 ) -
     &                        Rc*DCOS(Temp)
            DeltaLat = DASIN( Height*SinTemp / R(4) )
            GeoCnLat = Delta - DeltaLat
            i = i + 1
          ENDDO

        IF ( i.GE.10 ) THEN
            Write(*,*) 'ATMOS latitude iteration did NOT converge '
          ENDIF
*
```

```
      ALT = Height*REarthKm

* ------------- Determine Density based on altitude ----------------
      IF(ALT.GE.800D0) THEN
        H      = 130.8D0
        RHONOM =  4.262D-17
        RHO    = RHONOM*DEXP((800.0D0-ALT)/H)
       ELSEIF(ALT.GE.700D0) THEN
          H      = 105.3D0
          RHONOM =  1.216D-16
          RHO    = RHONOM*DEXP((700.0D0-ALT)/H)
         ELSEIF(ALT.GE.600D0) THEN
            H      = 91.0D0
            RHONOM =  3.818D-16
            RHO    = RHONOM*DEXP((600.0D0-ALT)/H)
           ELSEIF(ALT.GE.500D0) THEN
              H      = 81.9D0
              RHONOM =  1.316D-15
              RHO    = RHONOM*DEXP((500.0D0-ALT)/H)
             ELSEIF(ALT.GE.400D0) THEN
                H      = 73.2D0
                RHONOM =  5.192D-15
                RHO    = RHONOM*DEXP((400.0D0-ALT)/H)
               ELSEIF(ALT.GE.300D0) THEN
                  H      = 61.2D0
                  RHONOM =  2.653D-14
                  RHO    = RHONOM*DEXP((300.0D0-ALT)/H)
                 ELSEIF(ALT.GE.250D0) THEN
                    H      = 52.6D0
                    RHONOM =  7.316D-14
                    RHO    = RHONOM*DEXP((250.0D0-ALT)/H)
                   ELSEIF(ALT.GE.200D0) THEN
                      H      = 40.8D0
                      RHONOM =   2.706D-13
                      RHO    = RHONOM*DEXP((200.0D0-ALT)/H)
                     ELSEIF(ALT.GE.150D0) THEN
                        H      = 24.1D0
                        RHONOM =   2.141D-12
                        RHO    = RHONOM*DEXP((150.0D0-ALT)/H)
                       ELSEIF(ALT.GE.130D0) THEN
                          H      = 16.1D0
                          RHONOM =   8.484D-12
                          RHO    = RHONOM*DEXP((130.0D0-ALT)/H)
                         ELSE
                            H      =   8.06D0
                            RHONOM =   9.661D-11
                            RHO    = RHONOM*DEXP((110.0D0-ALT)/H)
                         ENDIF
      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                                SUBROUTINE CHEBY
*
*  This subroutine calculates a CHEBYCHEV expansion for the atmosphere.
*    Given an altitude above the Earth's surface, it will find the pressure and
*    density at that altitude using a Chebyshev polynomial.  Calculations are
*    accomplished in metric units, and the final answers are converted to
*    English units, as described below.
*    The model is only valid from 0 to 200 km (656,000 ft) altitude.
*
*  Algorithm     : Convert the altitude to km
*                  Assign the pressure coeff based on altitude
*                  Calculate the pressure
*                  Assign the density coeff based on altitude
*                  Calculate the denisty
*                  Convert to ENGLISH units
*
*  Author        : C2C Gandhi         USAFA       719-472-4109   28 Nov 1988
*                  Capt Dave Vallado  USAFA/DFAS  719-472-4109   28 Aug 1990
*
*  Inputs        :
*    Alt         - Altitude above earth's surface,         ft
*
*  Outputs       :
*    PAlt        - Pressure at altitude                lbf/in**2
*    RhoAlt      - Density at altitude                 lbm/ft**3
*
*  Locals        :
*    ..          - ..                                      .
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    None.
*
* ---------------------------------------------------------------------------

      SUBROUTINE CHEBY ( ALT, PALT,RHOALT )
        IMPLICIT NONE
        REAL*8 Alt,PAlt,RhoAlt

        REAL*8 Z, Z1, Sum, P0 , X, Nu, Part, LnR, R, Rho0, C(15),A(15)
        INTEGER k

* ------------------- Convert altitude to kilometers ------------------
        Z = Alt * 0.0003048D0
*
```

```
      IF (Z.le.80.0D0) THEN
* ------------- Chebychev model for altitudes of 80 km or less --------
* ----------- Define initial and zero altitude pressure constants -----
            SUM   =      0.0D0
            Z1    =     80.0D0
            P0    = 101325.0D0
* --------------- Define the pressure ratio coefficients --------------
            A(1)  = -11.385925D0
            A(2)  =  -5.6837011D0
            A(3)  =   0.052666476D0
            A(4)  =  -0.077884294D0
            A(5)  =  -0.11004083D0
            A(6)  =   0.017572339D0
            A(7)  =   0.0048546337D0
            A(8)  =   0.0017694805D0
            A(9)  =  -0.0018185298D0
            A(10) =  -0.0026635086D0
            A(11) =   0.0035685433D0
            A(12) =  -0.00082257517D0
            A(13) =  -0.0010363683D0
            A(14) =   0.00057053477D0
            A(15) =  -0.00019023078D0
         ELSE
* ------------- Chebychev model for altitudes of 80 to 200 km ---------
* --------- Define initial and zero altitude pressure constants -------
            SUM =      0.0D0
            Z1  =    200.0D0
            P0  = 101325.0D0
* ------------------- Define the pressure ratio coefficients ----------
            A(1)  = -24.475069D0
            A(2)  = -10.685861D0
            A(3)  =  2.2622605D0
            A(4)  =  0.63433398D0
            A(5)  = -0.27948959D0
            A(6)  = -0.31548574D0
            A(7)  =  0.090751361D0
            A(8)  =  0.18530467D0
            A(9)  = -0.095325843D0
            A(10) = -0.050214309D0
            A(11) =  0.045101378D0
            A(12) =  0.00889.7472D0
            A(13) = -0.018935899D0
            A(14) =  0.0035690621D0
            A(15) = -0.0063989880D0
         ENDIF

* ------------- Define X as a function of the altitude ratio ----------
      X     = 2.0D0 * Z/Z1 - 1.0D0

* --------------------- Define Nu as a function of X ------------------
      Nu    = 2.0D0 * X

* ----------- Define the Chebyshev Polynomials as functions of Nu -----
      C(2)  = Nu
      C(3)  = Nu*Nu - 2.0D0
      DO k= 4,15
          C(k)  = Nu * C(k-1) - C(k-2)
        ENDDO

* ------- Sum all parts of the Chebyshev expansion atmospheric model---
      DO k= 2,15
          PART = A(k) * C(k)
          SUM  = SUM + PART
        ENDDO

* ------------------- Solve for the pressure at altitude -------------
      LNR  = 0.5D0 * (A(1) + SUM)
      R    = DEXP(LNR)
      PALT = R * P0
*
```

```
      IF (Z.le.80.0D0) THEN
* ------------- Chebychev model for altitudes of 80 km or less --------
* ------------- Define initial and zero altitude density constants ----
          SUM   =  0.0D0
          Z1    =  80.0D0
          RHO0  =  1.2250D0
* ------------------ Define the density ratio coefficients ------------
          A(1)  = -10.960632D0
          A(2)  = -5.5717132D0
          A(3)  =  0.099116555D0
          A(4)  =  0.061044847D0
          A(5)  = -0.14304157D0
          A(6)  =  0.0029494088D0
          A(7)  =  0.0058789604D0
          A(8)  =  0.0020421324D0
          A(9)  =  0.0071033206D0
          A(10) = -0.0010314086D0
          A(11) =  0.0034100737D0
          A(12) =  0.0041764325D0
          A(13) = -0.0039151559D0
          A(14) =  0.0011227828D0
          A(15) = -0.0015751053D0
        ELSE

* ------------- Define initial and zero altitude density constants ----
          SUM = 0.0D0
          Z1  = 200.0D0
          RHO0  = 1.2250D0
* ------------- Chebychev model for altitudes of 80 to 200 km ---------
* ------------------ Define the density ratio coefficients ------------
          A(1)  = -25.415229D0
          A(2)  = -11.684380D0
          A(3)  =  1.8721406D0
          A(4)  =  0.81660876D0
          A(5)  = -0.093811118D0
          A(6)  = -0.30155735D0
          A(7)  = -0.077593291D0
          A(8)  =  0.21640168D0
          A(9)  = -0.034918422D0
          A(10) = -0.070126799D0
          A(11) =  0.036014616D0
          A(12) =  0.014951351D0
          A(13) = -0.021450283D0
          A(14) = -0.0012497995D0
          A(15) =  0.018421866D0
        ENDIF

* --------------- Define X as a function of the altitude ratio --------
      X     = 2.0D0 * Z/Z1 - 1.0D0

* ------------------------- Define Nu as a function of X --------------
      Nu    = 2.0D0 * X

* ------------ Define the Chebyshev Polynomials as functions of Nu ----
      C(2)  = Nu
      C(3)  = Nu*Nu - 2.0D0
      DO k=4,15
          C(k)  = Nu * C(k-1) - C(k-2)
        ENDDO

* ------- Sum all parts of the Chebyshev expansion atmospheric model --
      DO k= 2,15
          PART = A(k) * C(k)
          SUM  = SUM + PART
        ENDDO

* -------------------- Solve for the density at altitude --------------
      LNR  = 0.5D0 * (A(1) + SUM)
      R    = DEXP(LNR)
      RHOALT = R * RHO0

* ----- Convert pressure & density from metric units to English units--
* ------     (N/(m*m) ==> lbf/in**2   kg/m**3 ==> lbm/ft**3) ----------
      PAlt   = PAlt * 0.000145D0
      RhoAlt = RhoAlt * 0.062429507D0
    RETURN
    END
*
```

APPENDIX D

FORTRAN SOURCE CODE

MATHEMATICAL ROUTINES

```
*     ------------------------------------------------------------------       *
*                                                                              *
*                            Module - MATH.FOR                                 *
*                                                                              *
*  This file contains fundamental mathematical Subroutines and functions.     *
*                                                                              *
*      *****************  NOTICE OF GOVERNMENT ORIGIN  ****************       *
*                                                                              *
*   This software has been developed by an employee of the United States      *
*   Government at the United States Air Force Academy, and is therefore       *
*   a work of the United States, and is NOT subject to copyright protection   *
*   under the provisions of 17 U.S.C. 105.  ANY use of this work, or          *
*   inclusion in other works, must comply with the notice provisions of       *
*   17 U.S.C. 403.                                                             *
*                                                                              *
*      ****************************************************************       *
*                                                                              *
*  Author   : Capt Dave Vallado, USAFA Dept of Astronautics                   *
*                                                                              *
*  Coml  719-472-4109, Autovon 259-4109 / 4110                                *
*                                                                              *
*  Current : 30 Jan 91 Capt Dave Vallado          VERSION 3.0                 *
*                                                                              *
*  Changes : 25 Jan 91 Capt Dave Vallado                                      *
*                       Upgrade to LAHEY Ver 3.0 / FORTRAN 90                 *
*            20 Sep 90 Capt Dave Vallado          VERSION 2.1                 *
*                       Misc fixes to speed up                                *
*            20 Apr 90 Capt Dave Vallado          VERSION 2.0                 *
*                                                                              *
*            14 Nov 89 Capt Dave Vallado                                      *
*                                                                              *
*            24 Apr 89 Capt Dave Vallado                                      *
*                       Version 1.2                                           *
*            12 Feb 88 Capt Dave Vallado                                      *
*                       Standardized Format                                   *
*             8 Sep 88 Capt Dave Vallado                                      *
*                       Added Determinant and others                          *
*            30 Aug 88 Capt Dave Vallado                                      *
*                       Version 1.0                                           *
*                                                                              *
*     ------------------------------------------------------------------       *

* CONTENTS:
*
* ------------------------ Misc Functions ---------------------------
*
*  Function   Cot              ( XVal                              )
*
*  Function   Csc              ( XVal                              )
*
*  Function   Sec              ( XVal                              )
*
*  Function   DACOSH           ( XVal                              )
*
*
* ----------------------- Vector Operations -------------------------
*
*  Function   DOT              ( Vec1,Vec2                         )
*
*  Subroutine CROSS            ( Vec1,Vec2,                 OutVec )
*
*  Subroutine MAG              ( Vec                               )
*
*  Subroutine NORM             ( Vec,                       OutVec )
*
*  Subroutine ROT1             ( Vec,XVal,                  OutVec )
*
*  Subroutine ROT2             ( Vec,XVal,                  OutVec )
*
*  Subroutine ROT3             ( Vec,XVal,                  OutVec )
*
*  Subroutine ADDVEC           ( Vec1,Vec2,                 OutVec )
*
*  Subroutine ADD3VEC          ( Vec1,Vec2,Vec3,            OutVec )
*
*  Subroutine LNCOM1           ( A1,Vec,                    OutVec )
*
*  Subroutine LNCOM2           ( A1,A2,Vec1,Vec2, .         OutVec )
*
*  Subroutine LNCOM3           ( A1,A2,A3,Vec1,Vec2,Vec3,   OutVec )
*
*  Subroutine ANGLE            ( Vec1,Vec2,                 Theta  )
*
*
```

```
* -------------------------- Analytic routines ------------------------
*  Subroutine Quadratic           ( a,b,c               R1r,R1i,R2r,R2i  )
*
*  Subroutine Cubic               ( a,b,c,d             R1r,R1i,R2r,R2i,
*                                                       R3r,R3i          )
*
*  Subroutine Quartic             ( a,b,c,d,e           R1r,R1i,R2r,R2i,
*                                                       R3r,R3i,R4r,R4i  )
*
*
* ------------------------ Matrix Operations ------------------------
*
*  Subroutine MatMult             ( Mat1,Mat2,r1,c1,c2,Maxr1,Maxr2,Maxr3,
*                                   Maxc1,Maxc2,Maxc3,          Mat3    )
*
*  Subroutine MatAdd              ( Mat1,Mat2,r1,c1,Maxr1,Maxr2,Maxr3,
*                                   Maxc1,Maxc2,Maxc3,          Mat3    )
*
*  Subroutine MatTrans            ( Mat1,r1,c1,Maxr1,Maxr2,Maxc1,Maxc2,
*                                                               Mat2    )
*
*  Subroutine LUDeComp            ( LU,Index,                   Order   )
*
*  Subroutine LUBkSub             ( LU,Index,Order,             B       )
*
*  Subroutine MatInverse          ( Mat,Order,MaxRow,           MatInv  )
*
*  Subroutine PrintMat            ( Mat1,Row,Col,MaxRow,MaxCol          )
*
*  Function   Determinant         ( Order,                      Mat1    )
*
**
*
*  CONSTANTS:
*
*    Rad      = Radians per degree  = 57.29577951308230
*
*
```

```
* ------------------------------------------------------------------------------
*
*                              FUNCTION COT
*
*  This Function finds the Cotangent of an angle in radians.
*
*  Algorithm     : Find the tangent of the angle
*                  Check to avoid a divide by zero
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    XVal        - Angle to take Cotangent of                  rad
*
*  OutPuts       :
*    Cot         - Result
*
*  Locals        :
*    Temp        - Temporary Real variable
*    Infinity    - Large value to represent infinity
*
*  Coupling      :
*    None.
*
* ------------------------------------------------------------------------------

      REAL*8 FUNCTION Cot( XVal )
        IMPLICIT NONE
        REAL*8 XVal

* ----------------------------- Locals -------------------------------
        REAL*8 Temp,Infinity

* ------------------------ Implementation ---------------------------
        Infinity = 999999.9D0
        Temp     = DTAN( XVal )
        IF ( DABS(Temp).LE.0.000001D0 ) THEN
            Cot = Infinity
          ELSE
            Cot = 1.0D0 / Temp
          ENDIF

      RETURN
      END

*
```

```
* -----------------------------------------------------------------------------
*
*                              FUNCTION CSC
*
*  This Function finds the Cosecant of an angle in radians.
*
*  Algorithm     : Find the sine of the angle
*                  Check to avoid a divide by zero
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    XVal        - Angle to take Cosecant of                    rad
*
*  OutPuts       :
*    Csc         - Result
*
*  Locals        :
*    Temp        - Temporary Real variable
*    Infinity    - Large value to represent infinity
*
*  Coupling      :
*    None.
*
* -----------------------------------------------------------------------------

      REAL*8 FUNCTION Csc( XVal )
        IMPLICIT NONE
        REAL*8 XVal

* ----------------------------- Locals ------------------------------
        REAL*8 Temp,Infinity

* ----------------------- Implementation --------------------------
        Infinity = 999999.9D0
        Temp     = DSIN( XVal )
        IF ( DABS(Temp).LE.0.000001D0 ) THEN
            Csc = Infinity
          ELSE
            Csc = 1.0D0 / Temp
          ENDIF

      RETURN
      END

*
```

```
* -----------------------------------------------------------------------------
*
*                               FUNCTION SEC
*
*  This Function finds the secant of an angle in radians.
*
*  Algorithm     : Find the cosine of the angle
*                  Check to avoid a divide by zero
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    XVal        - Angle to take secant of                       rad
*
*  OutPuts       :
*    Sec         - Result
*
*  Locals        :
*    Temp        - Temporary Real variable
*    Infinity    - Large value to represent infinity
*
*  Coupling      :
*    None.
*
* -----------------------------------------------------------------------------

      REAL*8 FUNCTION Sec( XVal )
        IMPLICIT NONE
        REAL*8 XVal

* ----------------------------- Locals ------------------------------
        REAL*8 Temp,Infinity

* ----------------------- Implementation --------------------------
        Infinity = 999999.9D0
        Temp     = DCOS( XVal )
        IF ( DABS(Temp).LE.0.000001D0 ) THEN
            Sec = Infinity
          ELSE
            Sec = 1.0D0 / Temp
          ENDIF

      RETURN
      END
*
```

```
* ----------------------------------------------------------------------------
*
*                          FUNCTION DACOSH
*
*  This function evaluates the inverse hyperbolic cosine function.
*
*  Algorithm     : Check for an undefined value
*                  Calculate the answer using LOG base 10
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   28 Mar 1990
*
*  Inputs        :
*    XVal        - Value                                        1.0 to Infinity
*
*  OutPuts       :
*    DACosh      - Result                                       any real
*
*  Locals        :
*    Temp        - Temporary Real Value
*    Undefined   - Flag value for an undefined quantity
*
*  Coupling      :
*    None.
*
* ----------------------------------------------------------------------------

      REAL*8 FUNCTION DACOSH( XVal )
        IMPLICIT NONE
        REAL*8 XVal

* ----------------------------- Locals -------------------------------
        REAL*8 Temp,UnDefined

* ------------------------- Implementation   -------------------------
        Undefined= 999999.1D0

        IF (XVal**2-1.0D0.LT.0.0D0) THEN
            Temp= Undefined
            Write(*,*)  'Error in ArcCosh Function '
          ELSE
            Temp= DLOG( XVal + DSQRT( XVal*XVal - 1.0D0 ) )
          ENDIF

         DACOSH = Temp

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                              FUNCTION DOT
*
*  This Function finds the dot product of two vectors.
*
*  Algorithm     : Calculate the answer directly
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    Vec1        - Vector number 1
*    Vec2        - Vector number 2
*
*  OutPuts       :
*    Dot         - Result
*
*  Locals        :
*    None.
*
*  Coupling      :
*    None.
*
* ---------------------------------------------------------------------------

      REAL*8 FUNCTION DOT( Vec1,Vec2 )
        IMPLICIT NONE
        REAL*8 Vec1(4),Vec2(4)

* -----------------------  Implementation  --------------------------
        DOT= Vec1(1)*Vec2(1) + Vec1(2)*Vec2(2) + Vec1(3)*Vec2(3)

      RETURN
      END

* ---------------------------------------------------------------------------
*
*                              SUBROUTINE CROSS
*
*  This Subroutine crosses two vectors.
*
*  Algorithm     : Calculate each vector component
*                  Find the magnitude of the answer
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    Vec1        - Vector number 1
*    Vec2        - Vector number 2
*
*  OutPuts       :
*    OutVec      - Vector result of A x B
*
*  Locals        :
*    None.
*
*  Coupling      :
*    MAG         - Magnitude of a vector
*
* ---------------------------------------------------------------------------

      SUBROUTINE CROSS( Vec1,Vec2, OutVec )
        IMPLICIT NONE
        REAL*8 Vec1(4), Vec2(4), OutVec(4)

* -----------------------  Implementation  --------------------------
        OutVec(1)= Vec1(2)*Vec2(3) - Vec1(3)*Vec2(2)
        OutVec(2)= Vec1(3)*Vec2(1) - Vec1(1)*Vec2(3)
        OutVec(3)= Vec1(1)*Vec2(2) - Vec1(2)*Vec2(1)
        CALL MAG( OutVec )

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                              SUBROUTINE MAG
*
*  This Subroutine finds the magnitude of a vector.  The tolerance is set
*    for 0.000001, thus the 1.0D-12 for a squared test of underflows.
*
*  Algorithm     : Find the squared sum of the terms
*                  Check to be sure there is no SQRT of 0.0
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    Vec         - Vector
*
*  OutPuts       :
*    Vec(4)      - Answer stored in fourth component
*
*  Locals        :
*    Temp        - Temporary Real variable
*
*  Coupling      :
*    None.
*
* -----------------------------------------------------------------------------

      SUBROUTINE MAG( Vec )
        IMPLICIT NONE
        REAL*8 Vec(4)

* ----------------------------- Locals -------------------------------
        REAL*8 Temp

* ------------------------ Implementation ---------------------------
        Temp= Vec(1)**2 + Vec(2)**2 + Vec(3)**2
        IF (DABS(Temp).gt.1.0D-12) THEN
            Vec(4)= DSQRT( Temp )
          ELSE
            Vec(4)= 0.0D0
          ENDIF
      RETURN
      END

*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE NORM
*
*  This Subroutine calculates a unit vector given the original vector.  If a
*    zero vector is input, the output is set to zero.
*
*  Algorithm     : Find the magnitude of the input vector if not done
*                  Check if the magnitude is greater than zero
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   10 Feb 1989
*
*  Inputs        :
*    Vec         - Vector
*
*  OutPuts       :
*    OutVec      - Unit Vector
*
*  Locals        :
*    Small       - Tolerance factor
*    i           - Index
*
*  Constants     :
*    None.
*
*  Coupling      :
*    MAG         - Magnitude of a vector
*
* ---------------------------------------------------------------------------

      SUBROUTINE NORM( Vec, OutVec )
        IMPLICIT NONE
        REAL*8 Vec(4),OutVec(4)

* ----------------------------- Locals -------------------------------
        REAL*8 Small
        INTEGER i

* ------------------------ Implementation ---------------------------
        Small = 0.000001D0

        CALL MAG( Vec )
        IF ( Vec(4).GT.Small ) THEN
            DO i= 1,4
                OutVec(i)= Vec(i) / Vec(4)
            ENDDO
          ELSE
            DO i= 1,4
                OutVec(i)= 0.0D0
            ENDDO
          ENDIF
      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE ROT1
*
*  This Subroutine performs a rotation about the 1st axis.
*
*  Algorithm     : Store 3rd component for later use
*                  Calculate Sins and Cosine values to make more efficient
*                  Find the new vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    Vec         - Input vector
*    XVal        - Angle of rotation                             rad
*
*  OutPuts       :
*    OutVec      - Vector result
*
*  Locals        :
*    c           - Cosine of angle XVal
*    s           - Sine of angle XVal
*    Temp        - Temporary REAL value
*
*  Coupling      :
*    None.
*
* ---------------------------------------------------------------------------

      SUBROUTINE ROT1( Vec, XVal,  OutVec )
        IMPLICIT NONE
        REAL*8 Vec(4), XVal , OutVec(4)

* ----------------------------- Locals -------------------------------
        REAL*8 C,S,Temp

* ------------------------ Implementation ---------------------------
        Temp= Vec(3)
        c= DCos( XVal )
        s= DSin( XVal )

        OutVec(3)= c*Vec(3) - s*Vec(2)
        OutVec(2)= c*Vec(2) + s*Temp
        OutVec(1)= Vec(1)
        OutVec(4)= Vec(4)

      RETURN
      END

*
```

```
* -----------------------------------------------------------------------------
*
*                               SUBROUTINE ROT2
*
*  This Subroutine performs a rotation about the 2nd axis.
*
*  Algorithm     : Store 3rd component for later use
*                  Calculate Sine and Cosine values to make more efficient
*                  Find the new vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    Vec         - Input vector
*    XVal        - Angle of rotation                             rad
*
*  OutPuts       :
*    OutVec      - Vector result
*
*  Locals        :
*    c           - Cosine of angle XVal
*    s           - Sine of angle XVal
*    Temp        - Temporary REAL value
*
*  Coupling      :
*    None.
*
* ------------------------------------------------------------------

      SUBROUTINE ROT2( Vec, XVal, OutVec )
        IMPLICIT NONE
        REAL*8 Vec(4), XVal , OutVec(4)

* ---------------------------- Locals ------------------------------
        REAL*8 C,S,Temp

* ------------------------ Implementation --------------------------
        Temp= Vec(3)
        c= DCos( XVal )
        s= DSin( XVal )

        OutVec(3)= c*Vec(3) + s*Vec(1)
        OutVec(1)= c*Vec(1) - s*Temp
        OutVec(2)= Vec(2)
        OutVec(4)= Vec(4)

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                          SUBROUTINE ROT3
*
*  This Subroutine performs a rotation about the 3rd axis.
*
*  Algorithm     : Store 2nd component for later use
*                  Calculate Sine and Cosine values to make more efficient
*                  Find the new vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    Vec         - Input vector
*    XVal        - Angle of rotation                              rad
*
*  OutPuts       :
*    OutVec      - Vector result
*
*  Locals        :
*    c           - Cosine of the angle XVal
*    s           - Sine of the angle XVal
*    Temp        - Temporary REAL value
*
*  Coupling      :
*    None.
*
* ------------------------------------------------------------------------

      SUBROUTINE ROT3( Vec, XVal, OutVec )
        IMPLICIT NONE
        REAL*8 Vec(4), XVal , OutVec(4)

* ----------------------------- Locals -------------------------------
        REAL*8 C,S,Temp

* ------------------------ Implementation ---------------------------
        Temp= Vec(2)
        c= DCos( XVal )
        s= DSin( XVal )

        OutVec(2)= c*Vec(2) - s*Vec(1)
        OutVec(1)= c*Vec(1) + s*Temp
        OutVec(3)= Vec(3)
        OutVec(4)= Vec(4)

      RETURN
      END

*
```

```
* ------------------------------------------------------------------------------
*
*                             SUBROUTINE ADDVEC
*
*  This Subroutine adds two vectors.
*
*  Algorithm     : Loop to find each component
*                  Find the magnitude of the vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    Vec1        - Vector number 1
*    Vec2        - Vector number 2
*
*  OutPuts       :
*    OutVec      - Vector result of Vec1 x Vec2
*
*  Locals        :
*    i           - Index
*
*  Coupling      :
*    MAG         - Magnitude of a vector
*
* ------------------------------------------------------------------------------

      SUBROUTINE ADDVEC( Vec1,Vec2, OutVec )
        IMPLICIT NONE
        REAL*8 Vec1(4),Vec2(4),OutVec(4)

* ---------------------------- Locals ------------------------------
        INTEGER i

* ------------------------ Implementation --------------------------
        DO i= 1,3
            OutVec(i)= Vec1(i) + Vec2(i)
        ENDDO
        CALL MAG( OutVec )

      RETURN
      END

* ------------------------------------------------------------------------------
*
*                             SUBROUTINE ADD3VEC
*
*  This Subroutine adds three vectors.
*
*  Algorithm     : Loop to find each component
*                  Find the magnitude of the vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    Vec1        - Vector number 1
*    Vec2        - Vector number 2
*    Vec3        - Vector number 3
*
*  OutPuts       :
*    OutVec      - Vector result of Vec1 + Vec2 + Vec3
*
*  Locals        :
*    i           - Index
*
*  Coupling      :
*    MAG         - Magnitude of a vector
*
* ------------------------------------------------------------------------------

      SUBROUTINE ADD3VEC( Vec1,Vec2,Vec3, OutVec )
        IMPLICIT NONE
        REAL*8 Vec1(4),Vec2(4),Vec3(4), OutVec(4)

* ---------------------------- Locals ------------------------------
        INTEGER i

* ------------------------ Implementation --------------------------
        DO i= 1,3
            OutVec(i)= Vec1(i) + Vec2(i) + Vec3(i)
        ENDDO
        CALL MAG( OutVec )

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE LNCOM1
*
*  This Subroutine calculates the linear combination of a vector
*    multiplied by a constant.
*
*  Algorithm     : Loop to find each combination
*                  Find the magnitude of the vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   21 Aug 1988
*
*  Inputs        :
*    A           - constant
*    Vec         - Vector
*
*  OutPuts       :
*    OutVec      - Vector result of A * Vec
*
*  Locals        :
*    i           - Index
*
*  Coupling      :
*    MAG         - Magnitude of a vector
*
* ---------------------------------------------------------------------------

      SUBROUTINE LNCOM1( A,Vec,  OutVec )
        IMPLICIT NONE
        REAL*8 A,Vec(4),OutVec(4)

* ----------------------------- Locals -------------------------------
        INTEGER i

* ------------------------ Implementation --------------------------
        DO i= 1,3
            OutVec(i)= A*Vec(i)
        ENDDO
        CALL MAG( OutVec )

      RETURN
      END
*
```

```
* ----------------------------------------------------------------------------
*
*                               SUBROUTINE LNCOM2
*
*  This Subroutine calculates the linear combination of two vectors
*    multiplied by two constants.
*
*  Algorithm     : Loop to find each combination
*                  Find the magnitude of the vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    A1          - constant number 1
*    A2          - constant number 2
*    Vec1        - Vector number 1
*    Vec2        - Vector number 2
*
*  OutPuts       :
*    OutVec      - Vector result of A1*Vec1 + A2*Vec2
*
*  Locals        :
*    i           - Index
*
*  Coupling      :
*    MAG         - Magnitude of a vector
*
* ----------------------------------------------------------------------------

      SUBROUTINE LNCOM2( A1,A2,Vec1,Vec2, OutVec )
        IMPLICIT NONE
        REAL*8 A1,A2,Vec1(4),Vec2(4),OutVec(4)

* ----------------------------- Locals -------------------------------
        INTEGER i

* ------------------------ Implementation ---------------------------
        DO i= 1,3
            OutVec(i)= A1*Vec1(i) + A2*Vec2(i)
        ENDDO
        CALL MAG( OutVec )

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                            SUBROUTINE LNCOM3
*
*  This Subroutine calculates the linear combination of three vectors
*    multiplied by three different constants.
*
*  Algorithm     : Loop to find each combination
*                  Find the magnitude of the vector
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    A1          - constant number 1
*    A2          - constant number 2
*    A3          - constant number 3
*    Vec1        - Vector number 1
*    Vec2        - Vector number 2
*    Vec3        - Vector number 3
*
*  OutPuts       :
*    OutVec      - Vector result of A1*Vec1 + A2*Vec2 + A3*Vec3
*
*  Locals        :
*    i           - Index
*
*  Coupling      :
*    MAG         - Magnitude of a vector
*
* ---------------------------------------------------------------------------

      SUBROUTINE LNCOM3( A1,A2,A3,Vec1,Vec2,Vec3,OutVec )
        IMPLICIT NONE
        REAL*8 A1,A2,A3,Vec1(4),Vec2(4),Vec3(4),OutVec(4)

* ---------------------------- Locals -------------------------------
        INTEGER i

* ----------------------- Implementation ---------------------------
        DO i= 1,3
            OutVec(i)= A1*Vec1(i) + A2*Vec2(i) + A3*Vec3(i)
        ENDDO
        CALL MAG( OutVec )

      RETURN
      END

*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE ANGLE
*
*  This Subroutine calculates the angle between two vectors.  The output is
*    set to 999999.1 to indicate an undefined value.  Be SURE to check for
*    this at the output phase.  The tolerance is set for 0.000001, thus the
*    1.0D-12 for a squared test of divide by zero.
*
*  Algorithm     : Check the denominator for a divide by zero
*                  Check for exactly 1.0 or -1.0 to avoid ArcCosine problems
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    Vec1        - Vector number 1
*    Vec2        - Vector number 2
*
*  OutPuts       :
*    Theta       - Angle between the two vectors              rad
*
*  Locals        :
*    Temp        - Temporary REAL variable
*
*  Constants      :
*    Undefined   - Undefined flag for a variable
*
*  Coupling      :
*    DOT         - Dot Product of two vectors
*
* ---------------------------------------------------------------------------

      SUBROUTINE ANGLE ( Vec1,Vec2,  Theta )
        IMPLICIT NONE
        REAL*8 Vec1(4),Vec2(4),Theta,Temp

* ----------------------------- Locals ------------------------------
        EXTERNAL DOT
        REAL*8   Undefined,Dot

* ------------------------ Implementation --------------------------
        Undefined = 999999.1D0

        IF ( Vec1(4)*Vec2(4).GT.1.0D-12 ) THEN
            Temp = DOT(Vec1,Vec2) / (Vec1(4)*Vec2(4))
            IF ( DABS(Temp).gt.1.0D0 ) THEN
                Temp = DSIGN( 1.0D0,Temp )
              ENDIF
            Theta = DACOS( Temp )
          ELSE
            Theta = Undefined
          ENDIF

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                              SUBROUTINE QUADRATIC
*
*  This subroutine solves for the two roots of a quadratic equation.  There are
*    no restrictions on the coefficients, and imaginary results are passed
*    out as separate values.  The general form is y = ax2 + bx + c.
*
*  Algorithm     : Initialize all values
*                  Find discriminate
*                  Use discriminate value to separate the root calculations
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  10 Jan 1991
*
*  Inputs        :
*    a           - Coefficient of x squared term
*    b           - Coefficient of x term
*    c           - Constant
*
*  OutPuts       :
*    R1r         - Real portion of Root 1
*    R1i         - Imaginary portion of Root 1
*    R2r         - Real portion of Root 2
*    R2i         - Imaginary portion of Root 2
*
*  Locals        :
*    Discrim     - Discriminate b2 - 4ac
*
*  Constants     :
*    None.
*
*  Coupling      :
*    None.
*
*  References    :
*    Escobal     pg. 433-434
*
* ------------------------------------------------------------------------

      SUBROUTINE Quadratic( a,b,c,  R1r,R1i,R2r,R2i )
      IMPLICIT NONE
      REAL*8 a,b,c, R1r,R1i,R2r,R2i

* ----------------------------- Locals -------------------------------
      REAL*8 Discrim

* --------------------------   Initialize   --------------------------
        R1r= 0.0D0
        R1i= 0.0D0
        R2r= 0.0D0
        R2i= 0.0D0

        Discrim= b*b - 4.0D0*a*c

* ------------------------- Real roots ----------------------------
        IF (Discrim.gt.0.0D0) THEN
            R1r= ( -b + DSQRT(Discrim) ) / ( 2.0D0*a )
            R2r= ( -b - DSQRT(Discrim) ) / ( 2.0D0*a )
          ELSE
* ------------------------ Complex roots ---------------------------
            R1r= -b / ( 2.0D0*a )
            R2r= R1r
            R1i=  DSQRT(-Discrim) / ( 2.0D0*a )
            R2i= -DSQRT(-Discrim) / ( 2.0D0*a )
          ENDIF

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                            SUBROUTINE CUBIC
*
*  This subroutine solves for the three roots of a cubic equation.  There are
*    no restrictions on the coefficients, and imaginary results are passed
*    out as separate values.  The general form is y = ax3 + bx2 + cx + d.  Note
*    that R1i will ALWAYS be ZERO since there is ALWAYS at least one REAL root.
*
*  Algorithm     : Initialize variables
*                  Find correct coeffcients for the form of solution
*                  IF Delta is positive
*
*                      IF Delta is zero
*
*                         else
*                           find answers where Delta is negative
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  10 Jan 1991
*
*  Inputs        :
*    a           - Coefficient of x cubed term
*    b           - Coefficient of x squared term
*    c           - Coefficient of x term
*    d           - Constant
*
*  OutPuts       :
*    R1r         - Real portion of Root 1
*    R1i         - Imaginary portion of Root 1
*    R2r         - Real portion of Root 2
*    R2i         - Imaginary portion of Root 2
*    R3r         - Real portion of Root 3
*    R3i         - Imaginary portion of Root 3
*
*  Locals        :
*    Temp1       - Temporary value
*    Temp2       - Temporary value
*    Root1       - Temporary value of the root
*    Root2       - Temporary value of the root
*    Root3       - Temporary value of the root
*    P           - Coefficient of x squared term where x cubed term is 1.0
*    Q           - Coefficient of x term where x cubed term is 1.0
*    R           - Coefficient of constant term where x cubed term is 1.0
*    Delta       - Discriminator for use with Cardans formula
*    E0          - Angle holder for trigonometric solution
*    Phi         - Angle used in trigonometric solution
*    CosPhi      - Cosine of Phi
*    SinPhi      - Sine of Phi
*    Small       - Tolerance factor
*    OneThird    - 1.0/3.0
*
*  Constants     :
*    Rad         - Radians per degree
*
*  Coupling      :
*    ATAN2       - Arctangent including check for 180-360 deg
*    POWER       - Raise a number to a power
*
*  References    :
*    Escobal     pg. 430-433
*
* ---------------------------------------------------------------------------
*
```

```
      SUBROUTINE Cubic ( a,b,c,d, R1r,R1i,R2r,R2i,R3r,R3i )
      IMPLICIT NONE
      REAL*8 a,b,c,d, R1r,R1i,R2r,R2i,R3r,R3i

* ------------------------------ Locals ------------------------------
      REAL*8 templ, temp2, Root1, Root2, Root3, P, Q, R, Delta,
     &       E0, CosPhi, SinPhi, Phi, OneThird, Rad, Small

* -------------------------- Initialize  --------------------------
        Rad      = 57.29577951308230D0
        OneThird = 1.0D0/3.0D0
        Small= 0.000001D0
        R1r  = 0.0D0
        R1i  = 0.0D0
        R2r  = 0.0D0
        R2i  = 0.0D0
        R3r  = 0.0D0
        R3i  = 0.0D0
        Root1= 0.0D0
        Root2= 0.0D0
        Root3= 0.0D0

* ---------------- Force coefficients into std form ------------------
        P= B/A
        Q= C/A
        R= D/A

        a= OneThird*( 3.0D0*Q - P*P )
        b= (1.0D0/27.0D0)*( 2.0D0*P**3 - 9.0D0*P*Q + 27.0D0*R )

        Delta= (a**3/27.0D0) + (b*b/4.0D0)

* ----------------------- Use Cardans formula ------------------------
        IF (Delta.gt.Small) THEN
            Templ= (-b*0.5D0)+DSQRT(Delta)
            Temp2= (-b*0.5D0)-DSQRT(Delta)
            IF (DABS(Templ).gt.Small) THEN
                Templ= DSIGN(1.0D0,Templ)*( DSIGN(1.0D0,Templ)*Templ )
     &                       **OneThird
              ENDIF
            IF (DABS(Temp2).gt.Small) THEN
                Temp2= DSIGN(1.0D0,Temp2)*( DSIGN(1.0D0,Temp2)*Temp2 )
     &                       **OneThird
              ENDIF
            Root1= Templ + Temp2
            Root2= -0.5D0*(Templ + Temp2)
            Root3= -0.5D0*(Templ + Temp2)
            R2i  = -0.5D0*DSQRT( 3.0D0 )*(Templ - Temp2)
            R3i  = -R2i
          ELSE
* -------------------- Evaluate zero point ---------------------------
            IF (DABS( Delta ).lt.Small)  THEN
                IF (DABS(b).gt.Small) THEN
                    Root1= -DSIGN(1.0D0,b)*2.0D0*
     &                       ( DSIGN(1.0D0,b)*b/2.0D0 )**OneThird
                    Root2=  DSIGN(1.0D0,b)* ( DSIGN(1.0D0,b)*b/2.0D0 )
     &                           **OneThird
                    Root3= Root2
                  ENDIF
* else let them be 0.0D0 since b is 0.0D0
              ELSE
* -------------------- Use trigonometric identities ------------------
                E0     = 2.0D0*DSQRT(-a*OneThird)
                CosPhi= (-b/(2.0D0*DSQRT(-a**3/27.0D0)) )
                SinPhi= DSQRT( 1.0D0-CosPhi**2 )
                Phi    = DATAN2( SinPhi,CosPhi )
                Root1 = E0*DCos( Phi*OneThird )
                Root2 = E0*DCos( Phi*OneThird + 120.0D0/Rad )
                Root3 = E0*DCos( Phi*OneThird + 240.0D0/Rad )
              ENDIF
          ENDIF

         R1r= Root1 - P*OneThird
         R2r= Root2 - P*OneThird
         R3r= Root3 - P*OneThird
       RETURN
       END
*
```

```
* ------------------------------------------------------------------------
*
*                           SUBROUTINE QUARTIC
*
*  This subroutine solves for the four roots of a quartic equation.  There are
*    no restrictions on the coefficients, and imaginary results are passed
*    out as separate values.  The general form is y = ax4 + bx3 + cx2 + dx + e.
*
*  Algorithm     :
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109  10 Jan 1991
*
*  Inputs        :
*    a           - Coeficient of x fourth term
*    b           - Coefficient of x cubed term
*    c           - Coefficient of x squared term
*    d           - Coefficient of x term
*    e           - Constant
*
*  OutPuts       :
*    R1r         - Real portion of Root 1
*    R1i         - Imaginary portion of Root 1
*    R2r         - Real portion of Root 2
*    R2i         - Imaginary portion of Root 2
*    R3r         - Real portion of Root 3
*    R3i         - Imaginary portion of Root 3
*    R4r         - Real portion of Root 4
*    R4i         - Imaginary portion of Root 4
*
*  Locals        :
*    Temp1       - Temporary value
*    Temp2       - Temporary value
*    Root1       - Temporary value of the root
*    Root2       - Temporary value of the root
*    Root3       - Temporary value of the root
*    s           - alternate variable
*    h           - Temporary value
*    hSqr        - h squared
*    hCube       - h Cubed
*    P           - Term in auxillary equation
*    Q           - Term in auxillary equation
*    R           - Term in auxillary equation
*    Delta       - Discriminator for use with Cardans formula
*    E0          - Angle holder for trigonometric solution
*    Phi         - Angle used in trigonometric solution
*    CosPhi      - Cosine of Phi
*    SinPhi      - Sine of Phi
*    Small       - Tolerance factor
*    OneThird    - 1.0/3.0
*    RPrime      - Values of roots before final work
*    Temp        - Temporary variable in finding MAX RPrime
*    Eta         - Constant coefficient in quadratic solutions
*    Beta        - Constant coefficient in quadratic solutions
*
*  Constants     :
*    Rad         - Radians per degree
*
*  Coupling      :
*    ATAN2       - Arctangent including check for 180-360 deg
*    POWER       - Raise a number to a power
*
*  References    :
*    Escobal     pg. 430-433
*
* ------------------------------------------------------------------------
*
```

```
      SUBROUTINE Quartic( a,b,c,d,e, R1r,R1i,R2r,R2i,R3r,R3i,R4r,R4i )
      IMPLICIT NONE
      REAL*8 a,b,c,d,e, R1r,R1i,R2r,R2i,R3r,R3i,R4r,R4i
* ----------------------------- Locals ------------------------------
      REAL*8 Temp1, Temp2, Root1, Root2, Root3, s, h, P, Q, R, Delta,
     &       E0, OneThird, CosPhi, SinPhi, Phi, RPrime, hSqr, HCube,
     &       Eta, Beta, rad, Small

* --------------------------  Initialize  ---------------------------
        Rad      = 57.2957795130823OD0
        OneThird = 1.0/3.0
        Small= 0.000001D0
        R1r  = 0.0D0
        R1i  = 0.0D0
        R2r  = 0.0D0
        R2i  = 0.0D0
        R3r  = 0.0D0
        R3i  = 0.0D0
        R4r  = 0.0D0
        R4i  = 0.0D0
        Root1= 0.0D0
        Root2= 0.0D0
        Root3= 0.0D0

* ---------------- Force coefficients into std form -----------------
        b= B/A
        c= C/A
        d= D/A
        e= E/A

        H     = -b/4
        HSqr  = H**2
        HCube = H**3

        P=                   6.0*HSqr   + 3.0*b*h + c
        Q=         4.0*HCube + 3.0*b*HSqr + 2.0D0*c*h + d
        R= h*HCube +  b*HCube +    c*HSqr    +     d*h   + e

        a=  (1.0D0/ 3.0D0)*( -P*P-12.0D0*R )
        b=  (1.0D0/27.0D0)*( -2.0D0*P*P*P+72.0D0*P*R-27.0D0*Q*Q )
        s= -(2.0D0/ 3.0D0)*P

        Delta= (a**3/27.0D0) + (b*b/4.0D0)

        IF (DABS(Q).gt.Small) THEN
* ------------------------ Use Cardans formula ------------------------
            IF (Delta.gt.Small) THEN
                Temp1= (-b*0.5D0)+DSQRT(Delta)
                Temp2= (-b*0.5D0)-DSQRT(Delta)
                IF (DABS(Temp1).gt.Small) THEN
                    Temp1= DSIGN(1.0D0,Temp1)*( DSIGN(1.0D0,Temp1)*
     &                          Temp1 )**OneThird
                  ENDIF
                IF (DABS(Temp2).gt.Small) THEN
                    Temp2= DSIGN(1.0D0,Temp2)*( DSIGN(1.0D0,Temp2)*
     &                          Temp2 )**OneThird
                  ENDIF
                Root1= Temp1 + Temp2
                Root2= -0.5D0*(Temp1 + Temp2)
                Root3= -0.5D0*(Temp1 + Temp2)
                R2i   = -0.5D0*DSQRT( 3.0D0 )*(Temp1 - Temp2)
                R3i   = -R2i
              ELSE
* -------------------- Evaluate zero point --------------------------
                IF (DABS( Delta ).lt.Small)  THEN
                    IF (DABS(b).gt.Small) THEN
                        Root1= -DSIGN(1.0D0,b)*2.0D0*
     &                          ( DSIGN(1.0D0,b)*b/2.0D0 )**OneThird
                        Root2=  DSIGN(1.0D0,b)* ( DSIGN(1.0D0,b)*b/
     &                                 2.0D0 )**OneThird
                        Root3= Root2
                      ENDIF
* else let them be 0.0D0 since b is 0.0D0
                  ELSE
* -------------------- Use trigonometric identities ------------------
                    E0     = 2.0D0*DSQRT(-a*OneThird)
                    CosPhi= (-b/(2.0D0*DSQRT(-a**3/27.0D0)) )
                    SinPhi= DSQRT( 1.0D0-CosPhi**2 )
                    Phi    = DATAN2( SinPhi,CosPhi )
                    Root1 = E0*DCos( Phi*OneThird )
                    Root2 = E0*DCos( Phi*OneThird + 120.0D0/Rad )
                    Root3 = E0*DCos( Phi*OneThird + 240.0D0/Rad )
                  ENDIF
              ENDIF
*
```

```
* -------------------- Find largest value of root --------------------
            RPrime= Root1+s
            IF ((RPrime.lt.Root2+s).and.(DABS(R2i).lt.0.00001D0)) THEN
                RPrime= Root2+s
              ENDIF
            IF ((RPrime.lt.Root3+s).and.(DABS(R3i).lt.0.00001D0)) THEN
                RPrime= Root3+s
              ENDIF

* ---------- Evaluate coefficients of two resulting Quadratics --------
            IF (RPrime.gt.Small) THEN
                Eta = 0.5*( P + RPrime - Q/DSQRT(RPrime) )
                Beta= 0.5*( P + RPrime + Q/DSQRT(RPrime) )
              ELSE
                Eta = 0.5*P
                Beta= 0.5*P
              ENDIF

            CALL Quadratic( 1.0D0, DSQRT(RPrime),Eta,
     &                      R1r,R1i,R2r,R2i )
            CALL Quadratic( 1.0D0,-DSQRT(RPrime),Beta,
     &                      R3r,R3i,R4r,R4i )

          ELSE
* ---------- Case where solution reduces to a quadratic ---------------
            CALL Quadratic( 1.0,P,R,   R1r,R1i,R2r,R2i )
            R1r= DSQRT( R1r )
            R1i= DSQRT( R1i )
            R2r= DSQRT( R2r )
            R2i= DSQRT( R2i )
            R3r= -R1r
            R3i= -R1i
            R4r= -R2r
            R4i= -R2i
          ENDIF

        R1r= R1r + h
        R2r= R2r + h
        R3r= R3r + h
        R4r= R4r + h
      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                              SUBROUTINE MATMULT
*
*  This Subroutine multiplies two matricies together.
*
*  Algorithm     : Loop through the Rows
*                     Loop through the Cols
*                        Loop through an index
*                           Multiply and add up each cell value
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    Mat1        - Matrix number 1
*    Mat2        - Matrix number 2
*    r1          - Actual number of rows in Mat1
*    c1          - Actual number of cols in Mat1
*    c2          - Actual number of cols in Mat2
*    Maxr1       - Maximum number of rows for Mat1, the # declared in main
*    Maxr2       - Maximum number of rows for Mat2, the # declared in main
*    Maxr3       - Maximum number of rows for Mat3, the # declared in main
*    Maxc1       - Maximum number of cols for Mat1, the # declared in main
*    Maxc2       - Maximum number of cols for Mat2, the # declared in main
*    Maxc3       - Maximum number of cols for Mat3, the # declared in main
*
*  OutPuts       :
*    Mat3        - Matrix result of Mat1 * Mat2
*
*  Locals        :
*    row         - Row counter for Mat3
*    col         - Col counter for Mat3
*    ktr         - Additional counter
*
*  Coupling      :
*    None.
*
* ---------------------------------------------------------------------------

      SUBROUTINE MatMult ( Mat1,Mat2,r1,c1,c2,Maxr1,Maxr2,Maxr3,
     &                     MaxC1,MaxC2,MaxC3,  Mat3 )
        IMPLICIT NONE
        INTEGER r1, c1, c2, Maxr1, Maxr2, Maxr3, Maxc1, Maxc2, Maxc3
        REAL*8  Mat1(Maxr1,MaxC1), Mat2(Maxr2,MaxC2), Mat3(Maxr3,MaxC3)

* ----------------------------- Locals ------------------------------
        INTEGER Row, Col, ktr

* ------------------------- Implementation  -------------------------
        DO Row = 1 , r1
          DO Col = 1 , c2
             Mat3(Row,Col) = 0.0D0
             DO ktr = 1 , c1
                Mat3(Row,Col) = Mat3(Row,Col) +
     &                          Mat1(Row,ktr) * Mat2(ktr,Col)
             ENDDO
          ENDDO
        ENDDO
      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                           SUBROUTINE MATADD
*
*  This subroutine adds two matricies together.
*
*  Algorithm     : Loop through the Rows
*                     Loop through the Cols
*                        Add the matricies
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    Mat1        - Matrix number 1
*    Mat2        - Matrix number 2
*    r1          - Actual number of rows in Mat1
*    c1          - Actual number of cols in Mat1
*    Maxr1       - Maximum number of rows for Mat1, the # declared in main
*    Maxr2       - Maximum number of rows for Mat2, the # declared in main
*    Maxr3       - Maximum number of rows for Mat3, the # declared in main
*    Maxc1       - Maximum number of cols for Mat1, the # declared in main
*    Maxc2       - Maximum number of cols for Mat2, the # declared in main
*    Maxc3       - Maximum number of cols for Mat3, the # declared in main
*
*  OutPuts       :
*    Mat3        - Matrix result of Mat1 + Mat2 of size r1 x c1
*
*  Locals        :
*    row         - Row counter for Mat3
*    col         - Col counter for Mat3
*
*  Coupling      :
*    None.
*
*  References    :
*    None.
*
* ------------------------------------------------------------------------

      SUBROUTINE MatAdd ( Mat1,Mat2,r1,c1,Maxr1,Maxr2,Maxr3,
     &                    Maxc1,Maxc2,Maxc3,   Mat3 )
        IMPLICIT NONE
        INTEGER r1, c1, Maxr1, Maxr2, Maxr3, Maxc1, Maxc2, Maxc3
        REAL*8  Mat1(Maxr1,Maxc1), Mat2(Maxr2,Maxc2), Mat3(Maxr3,Maxc3)

* ----------------------------- Locals ------------------------------
        INTEGER Row, Col

* ------------------------- Implementation   ------------------------
        DO Row = 1 , r1
            DO Col = 1 , c1
                 Mat3(Row,Col) = Mat1(Row,Col) + Mat2(Row,Col)
              ENDDO
          ENDDO

      RETURN
      END
*
```

```
* ---------------------------------------------------------------------------
*
*                          SUBROUTINE MATTRANS
*
*  This subroutine finds the transpose of a matrix.
*
*  Algorithm     : Loop through the Rows
*                     Loop through the Cols
*                        Switch the rows for columns
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    Mat1        - Matrix number 1
*    r1          - Actual number of rows in Mat1
*    c1          - Actual number of cols in Mat1
*    Maxr1       - Maximum number of rows for Mat1, the # declared in main
*    Maxr2       - Maximum number of rows for Mat2, the # declared in main
*    Maxc1       - Maximum number of cols for Mat1, the # declared in main
*    Maxc2       - Maximum number of cols for Mat2, the # declared in main
*
*  OutPuts       :
*    Mat2        - Matrix result of transpose Mat1
*
*  Locals        :
*    row         - Row counter for Mat2
*    col         - Col counter for Mat2
*
*  Coupling      :
*    None.
*
* ---------------------------------------------------------------------------

      SUBROUTINE MatTrans ( Mat1,r1,c1,Maxr1,Maxr2,Maxc1,Maxc2,  Mat2 )
        IMPLICIT NONE
        INTEGER r1,c1, Maxr1, Maxr2, Maxc1, Maxc2
        REAL*8  Mat1(Maxr1,Maxc1), Mat2(Maxr2,Maxc2)

* ----------------------------- Locals ------------------------------
        INTEGER Row,Col

* ------------------------- Implementation    -----------------------
        DO Row = 1,r1
           DO Col = 1,c1
              Mat2(Col,Row) = Mat1(Row,Col)
           ENDDO
        ENDDO

      RETURN
      END
*
```

```
* -----------------------------------------------------------------------------
*
*                         SURBOUTINE LUDECOMP
*
*  This subroutine decomposes a matrix in to an LU form.  Note when used with
*    MatInverse, MaxRow is set to 10.  Also, Scalo is hardwired to 30.
*
*  Algorithm     :
*
*
*  Author        : Maj Tom Riggs      USAFA/DFAS  719-472-4109   27 Apr 1989
*                  Capt Dave Vallado USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    Order       - Order of matrix
*
*  OutPuts       :
*    LU          - LU decomposition matrix
*    Index       - Index vector for pivoting
*    MaxRow      - Maximum number of rows in the matricies
*
*  Locals        :
*    i           - Index
*    j           - Index
*    k           - Index
*    Imax        - Pivot row pointer
*    Scale       - Scale factor vector
*    Sum         - Temporary Variables
*    AMax        - Temporary Variables
*    Dum         - Temporary Variables
*    Small       - Tolerance
*
*  Coupling      :
*    None.
*
*  References    :
*    Numerical Recipes by Flannery
*
* -----------------------------------------------------------------------------
*
```

```
      SUBROUTINE LUDeComp( LU,Index,Order,MaxRow )
        IMPLICIT NONE
        INTEGER MaxRow, Order
        INTEGER Index(Order)
        REAL*8  LU(MaxRow,MaxRow)

* ------------------------------ Locals ------------------------------
        INTEGER i, j, k, imax
        REAL*8  Small, Scale(30), Sum, AMax, Dum

* -------------------------- Implementation  -------------------------
        Small = 0.000001
        IMax  = 0
        DO I = 1 , Order
            AMax = 0.0D0
            DO J = 1 , Order
                IF ( DAbs( LU(i,j) ).GT.AMax ) THEN
                    AMax = DAbs( LU(i,j) )
                  ENDIF
            ENDDO
            IF (DABS(AMax).le.Small) THEN
                Write(*,*) ' Singular Matrix '
              ENDIF
            Scale(i) = 1.0D0 / AMax
        ENDDO

        DO j = 1 , Order
            DO i = 1 , j - 1
                Sum = LU(i,j)
                DO k = 1 , i - 1
                    Sum = Sum - LU(i,k)*LU(k,j)
                ENDDO
                LU(i,j) = Sum
            ENDDO
            AMax = 0.0D0
            DO i = j , Order
                Sum = LU(i,j)
                DO k = 1 , j - 1
                    Sum = Sum - LU(i,k)*LU(k,j)
                ENDDO
                LU(i,j) =   Sum
                Dum = Scale(i)*DAbs(Sum)
                IF (Dum.ge.AMax) then
                    IMax = i
                    AMax = Dum
                  ENDIF
            ENDDO
            IF (j.ne.imax) then
                DO k = 1 , Order
                    Dum = LU(imax,K)
                    LU(imax,k) = LU(j,k )
                    LU(j,k) =   Dum
                ENDDO
                Scale(imax) = Scale(j)
              ENDIF
            Index(j) = Imax
            IF (DAbs( LU(j,j) ).lt.Small) then
                Write(*,*) ' Matrix is Singular '
              ENDIF
            IF (j.ne.Order) then
                Dum = 1.0D0 / LU(j,j)
                DO i = j + 1 , Order
                    LU(i,j) =  Dum*LU(i,j)
                ENDDO
              ENDIF
        ENDDO

      RETURN
      END
*
```

```
* -------------------------------------------------------------------------
*
*                          SUBROUTINE LUBkSUB
*
*  This subroutine finds the inverse of a matrix using LU decomposition.  Note,
*    when this is used by MatInverse, MaxRow is set at 10.
*
*  Algorithm     :
*
*
*  Author        : Maj Tom Riggs     USAFA/DFAS  719-472-4109   28 Apr 1989
*                  Capt Dave Vallado USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    Order       - Order of matrix
*    LU          - LU decomposition matrix
*    Index       - Index vector for pivoting
*    MaxRow      - Maximum number of rows in the matricies
*
*  OutPuts       :
*    B           - Solution Vector
*
*  Locals        :
*    i           - Index
*    j           - Index
*    I0          - Pointer to non-zero element
*    IPtr        - Pivot Row Pointer
*    Sum         - Temporary Variables
*
*  Coupling      :
*    None.
*
*  References    :
*    Numerical Recipes by Flannery
*
* -------------------------------------------------------------------------

      SUBROUTINE LUBkSub( LU,Index,Order,B,MaxRow )
        IMPLICIT NONE
        INTEGER MaxRow, Order
        INTEGER Index(Order)
        REAL*8  LU(MaxRow,MaxRow),B(MaxRow)

* ----------------------------- Locals ------------------------------
        INTEGER i,j,iptr,i0
        REAL*8  Sum

* ------------------------- Implementation   ------------------------
        I0 = 0
        DO i = 1 , Order
            IPtr = Index(i)
            Sum = B(IPtr)
            B(Iptr) = B(i)
            IF (I0.ne.0) THEN
                DO j = I0 , i - 1
                    Sum = Sum - LU(i,j)*B(j)
                ENDDO
              ELSE
                IF (Sum.ne.0.0D0) THEN
                    I0 = I
                  ENDIF
              ENDIF
            B(i)=  Sum
        ENDDO

        B(Order)= B(Order) / LU(Order,Order)

        DO i = (Order - 1),1, -1
            Sum = B(i)
            DO j = i + 1 , Order
               Sum = Sum - LU(i,j)*B(j)
            ENDDO
            B(i)= Sum / LU(i,i)
        ENDDO

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                        SUBROUTINE MATINVERSE
*
*  This subroutine finds the inverse of a matrix using LU decomposition.  Note
*    the MAXIMUM size matrix which may be inverted is a 10x10!!
*
*  Algorithm     :
*
*
*  Author        : Maj Tom Riggs      USAFA/DFAS  719-472-4109   28 Apr 1989
*                  Capt Dave Vallado USAFA/DFAS  719-472-4109   22 Mar 1990
*
*  Inputs        :
*    A           - Matrix to invert
*    Order       - Order of matrix
*    MaxRow      - Maximum number of rows for A, the # declared in main
*
*  OutPuts       :
*    AInv        - Inverted matrix
*
*  Locals        :
*    i           - Index
*    j           - Index
*    Index       - Index vector for pivoting
*    LU          - LU decomposition matrix
*    B           - Operational vector to form MatInv
*
*  Coupling      :
*    LUDecomp    - Finds LU decompisition of a matrix
*    LUBkSub     - Finds LU back substitute results for system
*
*  References    :
*    Numerical Recipes by Flannery
*
* ------------------------------------------------------------------------

      SUBROUTINE MatInverse( A,Order,MaxRow, AInv )
        IMPLICIT NONE
        INTEGER Order,MaxRow
        REAL*8 A(MaxRow,MaxRow), AInv(MaxRow,MaxRow)

* ----------------------------- Locals -------------------------------
        INTEGER MaxR
        PARAMETER (MaxR = 10)
        INTEGER i,j,Index(MaxR)
        REAL*8  Lu(MaxR,MaxR),B(MaxR)

* ------------------------- Implementation   -----------------------
        DO i = 1 , Order
            Index(i) = i
            DO j = 1 , Order
                LU(i,j) = A(i,j)
            ENDDO
        ENDDO

        CALL LUDeComp( LU, Index, Order, MaxR )

        DO j = 1 , Order
            DO i = 1 , Order
               IF (i.eq.j) THEN
                   B(i) = 1.0D0
                 ELSE
                   B(i) = 0.0D0
                 ENDIF
            ENDDO
            CALL LUBkSub( LU, Index, Order, B, MaxR )

            DO i = 1 , Order
               AInv(i,j) = B(i)
            ENDDO

        ENDDO

      RETURN
      END
*
```

```
* ------------------------------------------------------------------------
*
*                         SUBROUTINE PRINTMAT
*
*  This subroutine prints a matrix. The user should be aware of trying to print
*    matricies with more than about 6 columns.  Although the code will allow for
*    up to 10 columns as written, the editing specification may need to be
*    changed to a smaller value than the F12.7 to accomodate larger matricies.
*    You do NOT have to use all 10 spaces when printing a matrix.
*
*  Algorithm     : Write out the title for the matrix
*                  Loop through the rows and print out 1 row at a time
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   20 Sep 1990
*
*  Inputs        :
*    Matrix      - Matrix
*    Text        - Text describing the name of the matrix
*    row         - Actual number of rows in Matrix
*    col         - Actual number of cols in Matrix
*    MaxRow      - Maximum number of rows for Matrix, the # declared in main
*    MaxCol      - Maximum number of cols for Matrix, the # declared in main
*
*  OutPuts       :
*    None.
*
*  Locals        :
*    RowKtr      - Row counter for Matrix
*    ColKtr      - Col counter for Matrix
*
*  Coupling      :
*    None.
*
* ------------------------------------------------------------------------

      SUBROUTINE PrintMatrix( Matrix,Text,Row,Col,MaxRow,MaxCol )
        IMPLICIT NONE
        INTEGER MaxRow, MaxCol, Row, Col
        REAL*8  Matrix(MaxRow,MaxCol)
        CHARACTER*8 Text

* ---------------------------- Locals -------------------------------
        INTEGER RowKtr, ColKtr

* ------------------------- Implementation   ------------------------
        Write(*,*) Text
        DO RowKtr= 1 ,Row
            Write( *,20 ) (Matrix(RowKtr,ColKtr),ColKtr=1,Col)
        ENDDO

 20     FORMAT( 10(F12.7,1X) )
      RETURN
      END
*
```

```
* ----------------------------------------------------------------------------
*
*                          FUNCTION DETERMINANT
*
*  This function calcuates the determinant value using L-U decompisition.
*  The formula must have a NON-ZERO number in the 1,1 position.  If the
*  function senses a NON-ZERO number in row 1, it exchanges row1 for a row
*  WITH a NON-ZERO number.
*
*  Algorithm     :
*
*
*  Author        : Capt Dave Vallado  USAFA/DFAS  719-472-4109   12 Aug 1988
*
*  Inputs        :
*    Order       - Order of determinaent (# of rows)
*    Mat1        - Matrix to find determinant of
*    MaxRow      - Maximum number of rows for Matrix, the # declared in main
*
*  OutPuts       :
*    Determinant - Result
*
*  Locals        :
*    i           - Index
*    j           - Index
*    k           - Index
*    Temp        -
*    D           -
*    Sum         -
*    L           -
*    U           -
*    Small       - Tolarance for comparing to 0.0
*
*  Coupling      :
*    None.
*
*  References:
*    Marion      pg. 168-172, 126-127
*
* ----------------------------------------------------------------------------
*
```

```
      REAL*8 FUNCTION DETERMINANT( Order,Matl,MaxRow )
        IMPLICIT NONE
        INTEGER Order, MaxRow
        REAL*8 Matl( MaxRow,MaxRow )

* ----------------------------- Locals ------------------------------
        INTEGER MaxR
        PARAMETER (MaxR = 10)
        INTEGER i,j,k
        REAL*8 Temp,D,Sum,Small,L(MaxR,MaxR),U(MaxR,MaxR)

* ------------------------ Implementation ---------------------------
        Small= 0.0000001D0
        Sum  = 0.0D0

* ---------------- Switch a non zero row to the first row -------------
        IF ( DABS( Matl(1,1) ).LT.Small) THEN
            j= 1
            DO WHILE (j.LE.Order)
                IF ( DABS( Matl(j,1) ).GT.Small) THEN
                    DO k=1,Order
                        Temp= Matl(1,k)
                        Matl(1,k)= Matl(j,k)
                        Matl(j,k)= Temp
                    ENDDO
                    j= Order + 1
                  ENDIF
                j= j+1
             ENDDO
          ENDIF

        DO i= 1,Order
            L(i,1)= Matl(i,1)
        ENDDO
        DO j= 1,Order
            U(1,j)= Matl(1,j)/L(1,1)
        ENDDO
        DO j= 2,Order
            DO i= j,Order
                Sum= 0.0D0
                DO k= 1,j-1
                    Sum= Sum+L(i,k)*U(k,j)
                ENDDO
                L(i,j)= Matl(i,j)- Sum
            ENDDO
            U(j,j)= 1.0
            IF (j.NE.Order) THEN
                DO i= j+1,Order
                    Sum= 0.0D0
                    DO k= 1,j-1
                        Sum= Sum+L(j,k)*U(k,i)
                    ENDDO
                    U(j,i)= (Matl(j,i)-Sum)/L(j,j)
                ENDDO
              ENDIF
        ENDDO
        D= 1.0D0
        DO i= 1,Order
            D= D*L(i,i)
        ENDDO
        Determinant= D

      RETURN
      END
*
```

APPENDIX E

TEST CASES

Contents

*'s indicate the answer is given in the reference cited

MISC TIME Test Cases

JULIANDAY and INVJULIANDAY:
REF: Escobal pg 18-21

Ephemeris Values

JD	Year	Jan 1, 0 Hr	Mean		GST Calculated
2451544.5	2000				99.9677947
2451179.5	1999				100.2065061
2450814.4	1998				100.4452177
2450449.5	1997				100.6839293
2450083.5	1996				99.9369936
2449718.5	1995				100.1757054
2449353.5	1994				100.4144172
2448988.5	1993				100.6531291
2448622.5	1992				99.9061937
2448257.5	1991				100.1449058
2447892.5	1990				100.3836180
2447527.5	1989	6 42 29.7644	29.3590	100.62232886	100.6223302
2447161.5	1988	6 39 30.1642	30.0945	99.87539345	99.8753951
2446796.5	1987	6 40 27.1354	27.3855	100.11410594	100.1141075
2446431.5	1986				100.3528200
2446066.5	1985				100.5915325
2445700.5	1984	6 39 21.7168	22.7031	99.84459595	99.8445978
2445335.5	1983	6 40 18.9168	19.9310	100.08304553	100.0833105
2444970.5	1982				100.3220232
2444605.5	1981				100.5607361
2444239.5	1980	6 39 14.7690	15.2510	99.81354553	99.8138016

Date Day Mon Yr Hr Min Sec	Julian Date	Longitude	GST	LST
1 Jan 1900 0: 0: 0.0000	2415019.5000000	-104.883000	99.1981398	354.3151398
* 12 Oct 1962 10:15:30.0000	2437949.9274306	298.221300	174.3881886	112.6091886
1 Jan 1980 0: 0: 0.0000	2444239.5000000	-104.883000	99.8138016	354.9308016
* 1 Jan 1985 6:48: 0.0000	2446066.7833333	298.221300	202.8707893	141.0920893
1 Jan 1987 0: 0: 0.0000	2446796.5000000	-104.883000	100.1141075	355.2311075
31 Dec 1987 0: 0: 0.0000	2447160.5000000	-104.883000	98.8897478	354.0067478
1 Jan 1988 0: 0: 0.0000	2447161.5000000	-104.883000	99.8753951	354.9923951
1 Dec 1988 0: 0: 0.0000	2447496.5000000	0.000000	70.0672619	70.0672619
31 Dec 1988 0: 0: 0.0000	2447526.5000000	-104.883000	99.6366828	354.7536828
14 Jul 1989 0: 0: 0.0000	2447721.5000000	0.000000	291.8379187	291.8379187
1 Aug 1989 0: 0: 0.0000	2447739.5000000	-104.883000	309.5795713	204.6965713
17 Aug 1989 0: 0: 0.0000	2447755.5000000	-104.883000	325.3499291	220.4669291
17 Aug 1989 14: 0: 0.0000	2447756.0833333	-104.883000	175.9249077	71.0419077
17 Aug 1989 14:35: 0.0000	2447756.1076389	-104.883000	184.6988634	79.8158634
17 Aug 1989 14:35:59.9999	2447756.1083333	-104.883000	184.9495474	80.0665471
2 Oct 1989 0: 0: 0.0000	2447801.5000000	-104.883000	10.6897079	265.8067079
31 Dec 1989 0: 0: 0.0000	2447891.5000000	0.000000	99.3979706	99.3979706
1 Jan 1990 0: 0: 0.0000	2447892.5000000	-104.883000	100.3836180	355.5006180
1 Jan 1991 0: 0: 0.0000	2448257.5000000	-104.883000	100.1449058	355.2619058
1 Jan 1992 0: 0: 0.0000	2448622.5000000	-104.883000	99.9061937	355.0231937
1 Jan 1993 0: 0: 0.0000	2448988.5000000	-104.883000	100.6531291	355.7701291
1 Jan 1994 0: 0: 0.0000	2449353.5000000	-104.883000	100.4144172	355.5314172
1 Jan 1995 0: 0: 0.0000	2449718.5000000	-104.883000	100.1757054	355.2927054
1 Jan 1996 0: 0: 0.0000	2450083.5000000	-104.883000	99.9369936	355.0539936
1 Jan 1997 0: 0: 0.0000	2450449.5000000	-104.883000	100.6839293	355.8009293
1 Jan 1998 0: 0: 0.0000	2450814.5000000	-104.883000	100.4452177	355.5622177
1 Jan 1999 0: 0: 0.0000	2451179.5000000	-104.883000	100.2065061	355.3235061
1 Jan 2000 0: 0: 0.0000	2451544.5000000	-104.883000	99.9677947	355.0847947
2 Jan 2000 0: 0: 0.0000	2451545.5000000	-104.883000	100.9534420	356.0704420
2 oct 2000 0: 0: 0.0000	2451819.5000000	-104.883000	11.0208203	266.1378203

* Escobal pg. 18, 21 and 22

SITE-TRACK and RAZEL Test Cases

NOTICE the same data set is used for both SITE-TRACK tests and the RAZEL test cases. RAZEL is simply the inverse process of finding the range, azimuth, elevation, and rate terms from the vectors.

```
*1.  BMW  Appendix D.1,1

    Given:
      39.0070 deg  Latitude            504.68000  km     Range
    -104.8830 deg  Longitude           105.60000  deg    Azimuth
    7180.0000  ft  Altitude             30.70000  deg    Elevation
     317.0200      Universal Time        2.08000  km/s   Range rate
      2            Day                   0.05000  deg/s  Azimuth rate
     Sep           Month                 0.07000  deg/s  Elevation rate
     1970          Year

    Find:    i          j           k            magnitude
    RS = 0.2045751 -0.7510033  0.6262484     0.9990216   DU
    VS = 0.0441842  0.0120359  0.0000000     0.0457942   DU/TU

    R  = 0.2790794 -0.7751794  0.6374576     1.0417008   DU
    V  = 0.2634728 -0.1492353  0.0519525     0.3072265   DU/TU

2.  BMW  Appendix D.1,2

      37.8000 deg  Latitude            300.00000  km     Range
    -175.9000 deg  Longitude           315.00000  deg    Azimuth
       0.0000  ft  Altitude             45.00000  deg    Elevation
    1905.1500      Universal Time       -5.00000  km/s   Range rate
      8            Day                  -0.20000  deg/s  Azimuth rate
     Oct           Month                -0.30000  deg/s  Elevation rate
     1970          Year

    RS = -0.4806057  0.6284402  0.6095710   0.9987471
    VS = -0.0369734 -0.0282758  0.0000000   0.0465462

    R  = -0.4691328  0.6521522  0.6485385   1.0324680
    V  =  0.0186569 -0.3501725 -0.5839385   0.6811410

3.  BMW  Appendix D.1,3

      29.8000 deg  Latitude           1510.00000  km     Range
     -78.5000 deg  Longitude           180.00000  deg    Azimuth
      15.0000  ft  Altitude             45.00000  deg    Elevation
    2210.5750      Universal Time        4.50000  km/s   Range rate
      27           Day                   0.50000  deg/s  Azimuth rate
     Dec           Month                 0.53000  deg/s  Elevation rate
     1970          Year

    RS =  0.8558456 -0.1476259  0.4940560   0.9991779
    VS =  0.0086854  0.0503525  0.0000000   0.0510961

    R  =  1.0809849 -0.1864604  0.4319837   1.1789426
    V  =  0.8084630 -1.2700249  1.5558259   2.1649873

  ** Tests combination of azimuth and elevation since Az=180 and El=45  **
  **                 is the same as Az=0 and El = 135                    **

4.  BMW  Appendix D.1,4

       0.0000 deg  Latitude           6378.16500  km     Range
      80.0401 deg  Longitude           120.00000  deg    Azimuth
       0.0000  ft  Altitude             90.00000  deg    Elevation
       0.0000      Universal Time        0.00000  km/s   Range rate
      1            Day                   0.00000  deg/s  Azimuth rate
     Jan           Month                -0.10000  deg/s  Elevation rate
     1970          Year

    RS = -0.9999889 -0.0047078  0.0000000   1.0000000
    VS =  0.0002770 -0.0588329  0.0000000   0.0588336

    R  = -1.9999822 -0.0094157  0.0000000   2.0000044
    V  =  0.0062952 -1.3371525 -0.7040786   1.5112058

  ** Tests condition where satellite is directly overhead with small Del **
                     Tolerance check case for RAZEL
```

5. BMW Appendix D.1,5

Given:

45.7000 deg	Latitude	897.50000	km	Range
72.9000 deg	Longitude	201.70000	deg	Azimuth
3610.0000 ft	Altitude	76.70000	deg	Elevation
1024.3000	Universal Time	-0.57000	km/s	Range rate
15	Day	-0.75000	deg/s	Azimuth rate
Nov	Month	0.48000	deg/s	Elevation rate
1970	Year			

Find:	i	j	k	magnitude	
RS =	0.1588097	-0.6814766	0.7122471	0.9984622	DU
VS =	0.0400937	0.0093433	0.0000000	0.0411680	DU/TU
R =	0.1737448	-0.7983034	0.7892482	1.1359526	DU
V =	0.5975069	0.5823526	0.6294953	1.0451858	DU/TU

*6. USAFA Astro 451 Problem # 2

Given:

77.0000 deg	Latitude	35533.92100	km	Range
-68.0000 deg	Longitude	169.85700	deg	Azimuth
0.0000 ft	Altitude	61.88300	deg	Elevation
1801.0000	Universal Time	-0.23720	km/s	Range rate
1	Day	-0.00355	deg/s	Azimuth rate
Feb	Month	0.00433	deg/s	Elevation rate
1979	Year			

Find:	i	j	k	magnitude
RS =	0.2021307	-0.1003493	0.9709376	0.9968182
VS =	0.0059049	0.0118921	0.0000000	0.0132769
R =	3.6534009	-1.2975403	5.1773391	6.4680590
V =	-0.1535650	0.4116385	0.2048733	0.4847696

*7. USAFA Astro 321 Problem # 1

Given:

77.7000 deg	Latitude	3409.25300	km	Range
-68.5000 deg	Longitude	37.66050	deg	Azimuth
164.0000 ft	Altitude	31.10590	deg	Elevation
308.0000	Universal Time	-1.18340	km/s	Range rate
10	Day	-0.12671	deg/s	Azimuth rate
Jan	Month	0.02544	deg/s	Elevation rate
1986	Year			

Find:	i	j	k	magnitude
RS =	0.0080091	0.2135659	0.9736285	0.9968084
VS =	0.0125648	0.0004712	0.0000000	0.0125737
R =	0.2824806	-0.0709120	1.3206178	1.3523517
V =	0.7770328	-0.3392076	0.1526145	0.8621931

ELORB and RANDV Test Cases

Notice this data set may be used for both ELORB and RANDV tests.

*1. USAFA Astro 320 Handout Problem #1.1

```
            i            j            k
R =     1.1372844   -1.0534274   -0.8550194     1.7703625  DU
V =     0.6510489    0.4521008    0.0381088     0.7935440  DU/TU

p     =   1.9199998
a     =   1.9999997
e     =   0.1999999
i     =  29.9999997
Omega =  29.9999963
Argp  = 219.9999795
Nuo   =  65.0000223
M     =  45.5811951

U     =   Undefined
L     =   Undefined
Cappi =   Undefined

****        Test of Elliptical Inclined Orbit        ****
```

*2. USAFA Astro 320 Handout Problem #1.2

```
R =     1.0561942   -0.8950922   -0.0823703     1.3869106
V =    -0.5981066   -0.6293575    0.1468194     0.8805557

p     =   1.4849799
a     =   1.4999797
e     =   0.1000000
i     = 170.0000001
Omega = 299.9999875
Argp  =  25.0000239
Nuo   = 314.9999636
M     = 322.6859034

U     =   Undefined
L     =   Undefined
Cappi =   Undefined

****        Test of Elliptical Inclined Orbit        ****
```

*3. USAFA Astro 320 Handout Problem #1.3

```
R =    -0.4222777    1.0078857    0.7041832     1.3000100
V =    -0.5002738   -0.5415267    0.47[illegible]788     0.8770547

p     =   1.3000101
a     =   1.3000101
e     =   0.0000001
i     =  50.0000018
Omega =  80.0000026
Argp  =   Undefined
Nuo   =   Undefined
M     =  44.9999979

U     =  44.9999979
L     =   Undefined
Cappi =   Undefined

****        Test of Circular Inclined Orbit        ****
```

*4. USAFA Astro 320 Handout Problem #1.4

```
R =    -0.7309361   -0.6794646   -0.8331183     1.3000099
V =    -0.6724131    0.0341802    0.5620652     0.8770547

p     =   1.3000100
a     =   1.3000100
e     =   0.0000001
i     = 115.0002009
Omega = 200.0000000
Argp  =   Undefined
Nuo   =   Undefined
M     = 315.0000000

U     = 315.0000000
L     =   Undefined
Cappi =   Undefined

****        Test of Circular Inclined Orbit        ****
```

ELORB and RANDV Test Cases

*5. USAFA Astro 320 Handout Problem #1.5

R =	-3.5651640	-3.5651640	0.0000000	5.0419033
V =	0.3143612	-0.2555279	0.0000000	0.4051141

```
p     =   4.1280004
a     =   4.3000002
e     =   0.1999999
i     =   0.0000000
Omega =   Undefined
Argp  =   Undefined
Nuo   = 204.9999984
M     = 216.1768671

U     =   Undefined
L     =   Undefined
Cappi =  20.0000016
```

**** Test of Elliptical Equatorial Orbit ****

*6. USAFA Astro 320 Handout Problem #1.6

R =	4.4279958	0.3873994	-0.0000000	4.4449100
V =	0.0842152	-0.4585911	0.0000000	0.4662596

```
p     =   4.2570599
a     =   4.3000605
e     =   0.1000000
i     = 180.0000000
Omega =   Undefined
Argp  =   Undefined
Nuo   = 115.0000235
M     = 104.2906454

U     =   Undefined
L     =   Undefined
Cappi = 239.9999770
```

**** Test of Elliptical Equatorial Orbit ****

*7. USAFA Astro 320 Handout Problem #1.7

R =	0.9720220	2.0845079	0.0000000	2.3000000
V =	-0.5976017	0.2786662	0.0000000	0.6593805

```
p     =   2.3000002
a     =   2.3000002
e     =   0.0000001
i     =   0.0000000
Omega =   Undefined
Argp  =   Undefined
Nuo   =   Undefined
M     =  64.9999999

U     =   Undefined
L     =  64.9999999
Cappi =   Undefined
```

**** Test of Circular Equatorial Orbit ****

*8. USAFA Astro 320 Handout Problem #1.8

R =	-0.2004582	2.2912478	-0.0000000	2.3000000
V =	0.6568713	0.0574688	-0.0000000	0.6593804

```
p     =   2.2999998
a     =   2.2999998
e     =   0.0000001
i     = 180.0000000
Omega =   Undefined
Argp  =   Undefined
Nuo   =   Undefined
M     = 265.0000002

U     =   Undefined
L     = 265.0000002
Cappi =   Undefined
```

**** Test of Circular Equatorial Orbit ****

ELORB and RANDV Test Cases

*9. USAFA Astro 320 Handout Problem #1.9

```
R =     0.5916109   -1.2889359   -0.3738343    1.4666667
V =     1.1486347   -0.0808249   -0.1942733    1.1677485

p     =   2.2000003
a     =    Infinity
e     =   1.0000002
i     =  15.0000008
Omega =  35.0000066
Argp  = 199.9999965
Nuo   =  59.9999980
M     = 323.8301540

U     =   Undefined
L     =   Undefined
Cappi =   Undefined

****        Test of Parabolic Inclined Orbit        ****
```

*10. USAFA Astro 320 Handout Problem #1.10

```
R =    -1.0343646   -0.4814891    0.1735524    1.1540634
V =     0.1322278    0.7785322    1.0532856    1.3164373

p     =   2.2000002
a     =    Infinity
e     =   1.0000001
i     = 120.0000012
Omega = 210.0000011
Argp  =  34.9999971
Nuo   = 335.0000053
M     = 313.2492351

U     =   Undefined
L     =   Undefined
Cappi =   Undefined

****        Test of Parabolic Inclined Orbit        ****
```

*11. USAFA Astro 320 Handout Problem #1.11

```
R =     0.9163903    0.7005747   -1.3909623    1.8070286
V =     0.1712704    1.1036199   -0.3810377    1.1800424

p     =   2.4150303
a     =  -3.5000428
e     =   1.3000001
i     =  55.0000008
Omega =  94.9999992
Argp  = 215.0000065
Nuo   =  74.9999925
M     =  12.1085408

U     =   Undefined
L     =   Undefined
Cappi =   Undefined

****        Test of Hyperbolic Inclined Orbit        ****
```

*12. USAFA Astro 320 Handout Problem #1.12

```
R =    12.3160223   -7.0604653   -3.7883759   14.6930721
V =    -0.5902725    0.2165037    0.1628339    0.6494693

p     =   2.4151907
a     =  -3.5002757
e     =   1.3000001
i     = 165.0000006
Omega = 235.0000236
Argp  =  35.0000182
Nuo   = 230.0000046
M     = 170.1406418

U     =   Undefined
L     =   Undefined
Cappi =   Undefined

****        Test of Hyperbolic Inclined Orbit        ****
```

ELORB and RANDV Test Cases

*1. USAFA Astro 320 Handout Problem #1.1 Final

R =	-0.4395790	-0.8344110	-0.4611020	1.0498031
V =	0.8860850	-0.3656480	-0.1816190	0.9756180

```
p     =   1.0489991
a     =   1.0490002
e     =   0.0009997
i     =  28.5000027
Omega = 357.9999498
Argp  =  26.9945554
Nuo   = 220.0054910
M     = 220.0791812

U     =   Undefined
L     =   Undefined
Cappi =   Undefined
```

**** Example of a shuttle orbit ****

*2. USAFA Astro 320 Handout Problem #1.2 Final

R =	0.7764100	0.5236950	0.5407000	1.0813998
V =	-0.3986130	-0.2688680	0.8327940	0.9616279

```
p     =   1.0814006
a     =   1.0814006
e     =   0.0000008
i     =  90.0000093
Omega =  33.9999974
Argp  =   Undefined
Nuo   =   Undefined
M     =  30.0000075

U     =  30.0000075
L     =   Undefined
Cappi =   Undefined
```

**** Example Polar orbit, surveillance ****

*3. USAFA Astro 320 Handout Problem #1.3 Final

R =	-3.9752320	-1.0966930	0.6458080	4.1739996
V =	-0.0050220	-0.2347100	-0.4294930	0.4894673

```
p     =   4.1739975
a     =   4.1739975
e     =   0.0000009
i     =  62.9999994
Omega =  20.0000073
Argp  =   Undefined
Nuo   =   Undefined
M     = 170.0000023

U     = 170.0000023
L     =   Undefined
Cappi =   Undefined
```

**** Sample of a GPS orbit ****

*4. USAFA Astro 320 Handout Problem #1.4 Final

R =	56.2767910	-46.0649980	-31.1585420	80.3005354
V =	-0.3494800	0.2659280	0.1972370	0.4814108

```
p     =   4.8275047
a     =  -4.8344217
e     =   1.4137076
i     = 146.5000870
Omega = 279.9998432
Argp  =  86.9995231
Nuo   = 228.3303462
M     = 101.5948009

U     =   Undefined
L     =   Undefined
Cappi =   Undefined
```

**** Sample Comet orbit ****

ELORB and RANDV Test Cases

```
*5.  USAFA Astro 320 Handout Problem #1.5 Final

     R =      0.0858850     0.0601370     1.1983940     1.2029717
     V =     -0.5553150    -0.3888360     0.1058350     0.6861263

     p     =   0.6781607
     a     =   0.8390808
     e     =   0.4379285
     i     =  90.0000140
     Omega =  35.0000152
     Argp  = 269.9999740
     Nuo   = 175.0000189
     M     = 168.5231854

     U     =   Undefined
     L     =   Undefined
     Cappi =   Undefined

    ****            Sample ICBM trajectory            ****

*6.  USAFA Astro 320 Handout Problem #1.6 Final

     R =      4.6744710    -4.6744710     0.0000000     6.6107003
     V =      0.2750180     0.2750180     0.0000000     0.3889342

     p     =   6.6106958
     a     =   6.6106958
     e     =   0.0000007
     i     =   0.0000000
     Omega =   Undefined
     Argp  =   Undefined
     Nuo   =   Undefined
     M     = 315.0000000

     U     =   Undefined
     L     = 315.0000000
     Cappi =   Undefined

    ****      Sample Communication or Early Warning Satellite  ****

*1.  USAFA Astro 320 Handout Problem #1.6 variation

     R =     -2.5494956     3.6410571    -0.0000000     4.4449100
     V =      0.3550439     0.3022281    -0.0000000     0.4662596

     p     =   4.2570597
     a     =   4.3000604
     e     =   0.1000001
     i     = 180.0000000
     Omega =   Undefined
     Argp  =   Undefined
     Nuo   = 115.0000153
     M     = 104.2906221

     U     =   Undefined
     L     =   Undefined
     Cappi = 119.9999850


2.  BMW orbit #1 pg. 65

     R =      1.0606602     1.0606602    -0.0000000     1.5000000
     V =      0.4618802    -0.6928203     0.0000000     0.8326664

     p     =   1.5000000
     a     =   1.5625000
     e     =   0.2000000
     i     = 180.0000000
     Omega =   Undefined
     Argp  =   Undefined
     Nuo   = 269.9999886
     M     = 292.7645813

     U     =   Undefined
     L     =   Undefined
     Cappi =  45.0000114

          **  Note that i in BMW is redundant for this case  **
```

3. BMW orbit #2 pg. 66

R =	0.0000000	-1.2353675	1.2353675	1.7470735
V =	0.0000000	0.5773503	0.4140510	0.7104728

```
p     =   1.5000002
a     =   1.5625001
e     =   0.1999999
i     =  90.0000000
Omega = 270.0000000
Argp  = 179.9999769
Nuo   = 225.0000231
M     = 243.0447589

U     =   Undefined
L     =   Undefined
Cappi =   Undefined
```

** Note that u and l in BMW are redundant for this case **

4. BMW orbit #3 pg. 67

R =	0.3750000	0.6495191	-1.2990381	1.5000000
V =	-0.7071068	0.4082483	-0.0000000	0.8164966

```
p     =   1.5000001
a     =   1.5000001
e     =   0.0000001
i     =  59.9999985
Omega = 150.0000001
Argp  =   Undefined
Nuo   =   Undefined
M     = 270.0000009

U     = 270.0000009
L     =   Undefined
Cappi =   Undefined
```

** Note that l is redundant for this case **

5. USAFA Astro 320 Handout Problem #1.1 Extra

R =	-1.1000000	0.0000000	0.0000000	1.1000000
V =	-0.0045379	0.0100000	0.0000001	0.0109815

```
p     =   0.0001210
a     =   0.5500365
e     =   0.9998900
i     = 180.0000000
Omega =   Undefined
Argp  =   Undefined
Nuo   = 179.9971397
M     = 179.2286881

U     =   Undefined
L     =   Undefined
Cappi =   0.0028603
```

**** Test the accuracy of tolerances in program ****

6. USAFA Astro 320 Handout Problem #1.2 Extra

R =	0.0228098	-0.1302512	0.9943682	1.0031220
V =	-0.0110796	0.0678955	-0.4985516	0.5032755

```
p     =   0.0000068
a     =   0.5745511
e     =   0.9999941
i     =  87.6756823
Omega = 262.1614208
Argp  = 262.7135085
Nuo   = 180.0750475
M     = 259.9215100

U     =   Undefined
L     =   Undefined
Cappi =   Undefined
```

**** Test of Rectilinear Ellipse Orbit ****

GIBBS Test Cases

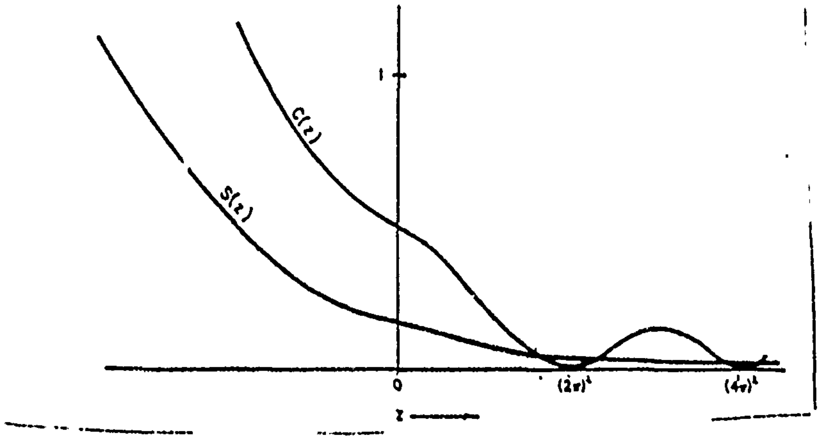
```
*1.  BMW  example pg. 115

    Given:
                 i         j         k              Find:    i           j            k         magnitude
    R1  =  0.000000  0.000000  1.000000  DU
    R2  =  0.000000 -0.700000 -0.800000  DU    V2  =  0.0000000  0.6986701 -0.6567445  0.9596101  DU\TU
    R3  =  0.000000  0.900000  0.500000  DU                       160.2405290


*2.  BMW  pg. 146 problem 2.13a

    R1  =  1.414225  0.000000  1.414202
    R2  =  1.810657  1.060669  0.310651        V2  = -0.0912544  0.3128428 -0.5336687  0.6253001
    R3  =  1.353540  1.414225 -0.646450                           45.0000138


3.  BMW  pg. 147 problem 2.13b

    R1  =  0.707113  0.000000  0.707101
    R2  = -0.894979  0.565681 -0.949642        V2  =  0.0000000  0.0000000  0.0000000  0.0000000
    R3  = -0.094979 -0.565681 -0.894977

    ***  Vectors are not coplanar ***

4.  BMW  pg. 147 problem 2.13c

    R1  =  1.000000  0.000000  0.000000
    R2  = -0.800000  0.600000  0.000000        V2  = -0.6000000 -0.8000000  0.0000000  1.0000000
    R3  =  0.800000 -0.600000  0.000000                          180.0000000


5.  BMW  pg. 147 problem 2.13d

    R1  =  0.207096  3.535528  1.207125
    R2  =  0.914201  4.949742  1.942347        V2  =  0.2509289  0.5018595  0.2475890  0.6132932
    R3  =  1.621305  6.363955  2.621344                            7.0617969


6.  BMW  pg. 147 problem 2.13e

    R1  =  1.000000  0.000000  0.000000
    R2  =  0.000000  1.000000  0.000000        V2  = -1.0000000  0.0000000  0.0000000  1.0000000
    R3  = -1.000000  0.000000  0.000000                           90.0000000


7.  BMW  pg. 147 problem 2.13f

    R1  =  7.000000  2.000000  0.000000 DU
    R2  =  1.000000  1.000000  0.000000 DU     V2  =  0.0000000  0.0000000  0.0000000  0.0000000
    R3  =  2.000000  7.000000  0.000000 DU

    ***  Orbit is not possible ***

8.  BMW  pg. 147 problem 2.13g

    R1  =  0.000000  2.700000  0.000000
    R2  =  2.970000  0.000000  0.000000        V2  =  0.0580259 -0.5802589  0.0000000  0.5831529
    R3  = -2.970000  0.000000  0.000000                          180.0000000


9.  USAFA Astro 451 Problem #3

    R1  =  0.000000  1.100000  0.000000
    R2  = -1.212992 -2.057288  1.212992        V2  =  0.1475475 -0.4985584 -0.1475475  0.5404637
    R3  =  0.000000 -3.300000  0.000000                          140.1776177


10.  USAFA Astro 321 Problem #2

    R1  =  0.000000  1.200000  0.000000
    R2  = -1.212992 -2.157288  1.212992        V2  =  0.1752110 -0.4974362 -0.1607665  0.5513507
    R3  =  0.000000 -3.400000  0.000000                          142.6525475

11.
    R1  =  1.200000  0.000000  0.000000
    R2  = -0.800000  0.000000  0.300122        V2  =  0.0000000  0.0000000  0.0000000  0.0000000
    R3  =  0.000000  0.900000  0.000000                            0.0000000

    ***  Vectors are not coplanar ***
```

HERRICK-GIBBS Test Cases

```
*1.  USAFA Astro 451 Problem #3

    Given:    i          j          k          Univ Time   Find:      i           j            k          magnitu

    R1  =  0.000000  1.100000  0.000000 DU  1201.00000
    R2  = -1.212992 -2.057288  1.212992 DU  1309.55480    V2  = -0.0000000 -0.8214425  0.0000000   0.82144
    R3  =  0.000000 -3.300000  0.000000 DU  1418.50960                      140.1776177

      ** Angles are too far apart  **


*2.  USAFA Astro 321 Problem #2

    R1  =  0.000000  1.200000  0.000000     1133.00000
    R2  = -1.212992 -2.157288  1.112992     1241.55450    V2  =  0.0038602 -0.7776137 -0.0035420   0.77763
    R3  =  0.000000 -3.400000  0.000000     1349.50900                      142.6525475

      ** Angles are too far apart  **


*3.  USAFA Astro 321 Problem #1 Fall 88

    R1  =  0.53618414  0.94382196  0.43658524  0.000000
    R2  =  0.51957094  0.95496111  0.43247339  0.169917  V2  = -0.7922407  0.5226298 -0.1980809   0.96954
    R3  =  0.50281293  0.96583641  0.42824155  0.339865                      1.0000002


*4.  USAFA Astro 321 Problem #2 Fall 88

    R1  =  0.53618414  0.94382196  0.43658524  0.000000
    R2  =  0.46030868  0.99185440  0.41714307  1.164760  V2  = -0.8148447  0.4778570 -0.2176481   0.96937
    R3  =  0.38176536  1.03437523  0.39533848  2.330378                      4.5000004


*5.  USAFA Astro 321 Problem #3 Fall 88

    R1  =  0.53618414  0.94382196  0.43658524   0.000000
    R2  = -1.04241241  0.71031097 -0.25535014  27.197046 V2  = -0.5086478 -0.5539836 -0.3345728   0.82314
    R3  = -0.65533617 -1.15356018 -0.53360418  62.369619                     90.0000004


6.

    R1  =  0.207096    3.535528    1.207125    3.7417
    R2  =  0.914201    4.949742    1.942347    5.3952     V2  =  6.5998156 13.1992165  6.6074758  16.16856
    R3  =  1.621305    6.363955    2.621344    7.0711                        7.0617969

7.
    R1  =  1.200000    0.000000    0.000000    0.0000
    R2  = -0.800000    0.000000    0.800122 1000.0000     V2  =  0.0000000  0.0000000  0.0000000   0.00000
    R3  =  0.000000    0.900000    0.000000 2000.0000                        0.0000000

    ***  Vectors are not coplanar ***
```

FindCandS:
REF: BMW chart on pg. 209

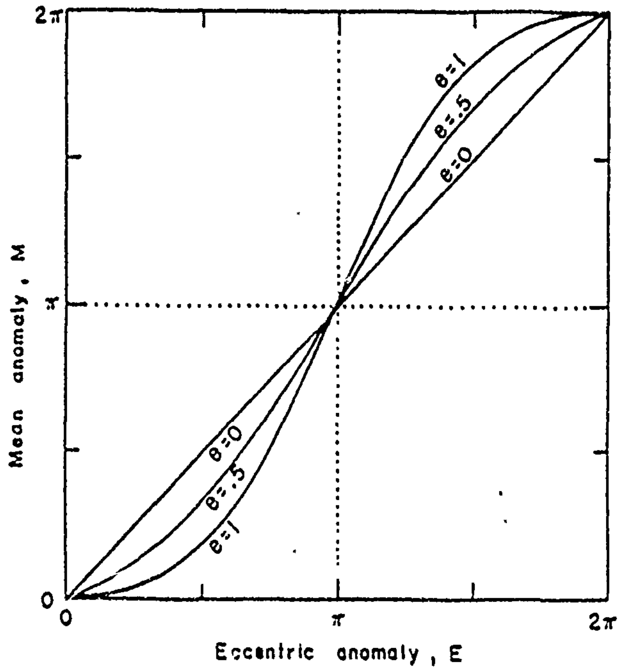
z	C	S
-39.47842	5.83559577	0.97444596
0.00000	0.50000000	0.16666667
0.57483	0.47650300	0.16194146
39.47842	0.00000000	0.02533029
50.00000	0.00589304	0.01799504



NewtonR:
REF: BMW chart on pg. 221

	Eccentricity	Mean Anomaly deg	Eccentric Anomaly deg	True Anomaly deg
*	0.04844	151.7425	153.002000	154.236000
	0.00000	90.0000	90.000000	90.000000
	0.49900	90.0000	115.751233	140.098510
	0.50000	90.0000	115.793623	140.177613
	0.51110	90.0000	116.261345	141.049998
	0.99900	90.0000	132.321144	178.867518
	1.00000	90.0000	132.346459	180.000000
	0.00000	270.0000	270.000000	270.000000
	0.49900	270.0000	244.248767	219.901490
	0.50000	270.0000	244.206377	219.822387
	0.51110	270.0000	243.738655	218.950002
	0.99900	270.0000	227.678856	181.132482
	1.00000	270.0000	227.653541	180.000000

* (Roy pg 85)



KEPLER Test Cases

*1. BMW example problem pg. 210

Ro =	1.0000000	0.0000000	0.0000000	1.0000000
Vo =	0.0000000	0.0000000	1.1000000	1.1000000
Dt =	2.00000 TU			
R =	-0.3206670	0.0000000	1.2364349	1.2773404
V =	-0.8799766	-0.0000000	-0.0373113	0.8807672
R =	-0.3187509	-0.0000000	1.2367231	1.2771398
V =	-0.8801729	0.0000000	-0.0359836	0.8809082

Iterations:

#	X	tn	dt	xn	c	s	z
1	1.58000	1.70506	1.22178	1.82140	0.15098	0.42304	1.97216
2	1.82140	2.00684	1.27860	1.81605	0.14614	0.39990	2.62083
3	1.81605	2.00000	1.27734	1.81605	0.14625	0.40044	2.60545

	Two-Body	Two-Body New	Perturbed New
p =	1.210000	1.2099960	1.2100000
a =	1.266000	1.2658188	1.2658228
e =	0.210000	0.2100003	0.2100000
i =	90.000000	90.0000000	90.0000000
Omega =	0.000000	0.0000000	0.0000000
Argp =	0.000000	359.9990766	359.9554007
Nu =	0.000000	104.5401464	104.4973687
M =	0.000000	80.4633328	80.4188239
U =	Undefined	Undefined	Undefined
L =	Undefined	Undefined	Undefined
Cappi =	Undefined	Undefined	Undefined

2. BMW Appendix D.3,1

Ro =	0.0000000	1.0000000	0.0000000	1.0000000
Vo =	0.0000000	0.0000000	1.0000000	1.0000000
Dt =	3.14159 TU			
R =	0.0000000	-1.0000000	0.0000000	1.0000000
V =	-0.0000000	-0.0000000	-1.0000000	1.0000000

1	3.04734	3.04734	1.00000	3.14159	0.10436	0.21489	9.28631
2	3.14159	3.14159	1.00000	3.14159	0.10132	0.20264	9.86960

p =	1.0000000	1.0000000
a =	1.0000000	1.0000000
e =	0.0000000	0.0000000
i =	90.0000000	90.0000000
Omega =	90.0000000	90.0000000
Argp =	Undefined	Undefined
Nu =	Undefined	Undefined
M =	0.0000000	179.9999998
U =	0.0000000	179.9999998
L =	Undefined	Undefined
Cappi =	Undefined	Undefined

** Tests first guess being too close **

*3. BMW Appendix D.3,2

```
Ro =       0.0000000      0.0000000     -0.5000000      0.5000000
Vo =       0.0000000      2.0000000      0.0000000      2.0000000
Dt = 1000000.00000 TU

R  =       0.0000000    181.7065561  16508.1362596  16509.1362596
V  =       0.0000000      0.0000606      0.0110064      0.0110066
```

1	181.60401	998308.06507	16490.50885	181.70661	0.16667	0.50000	0.00000
2	181.70661	1000000.95604	16509.14678	181.70656	0.16667	0.50000	0.00000
3	181.70656	1000000.00000	16509.13626	181.70656	0.16667	0.50000	0.00000

```
p     =   1.0000000        1.0000000
a     =    Infinity         Infinity
e     =   1.0000000        1.0000000
i     =  90.0000000       90.0000000
Omega =  90.0000000       90.0000000
Argp  = 270.0000000      270.0000000
Nu    =   0.0000000      179.3693656
M     =   0.0000000        0.0000000

U     =   Undefined        Undefined
L     =   Undefined        Undefined
Cappi =   Undefined        Undefined
```

** Tests first guess for a parabolic case **

4. BMW Appendix D.3,3

```
Ro =       0.3000000      1.0000000      0.0000000      1.0440307
Vo =       3.0000000      0.0000000      0.0000000      3.0000000
Dt =       5.00000 TU

R  =      13.9623306     -0.1172043      0.0000000     13.9628225
V  =       2.6781140     -0.2375952      0.0000000      2.6886328
```

1	1.07476	4.70954	13.17760	1.09680	0.24966	0.94757	-8.18323
2	1.09680	5.00844	13.97719	1.09620	0.25380	0.97132	-8.52232
3	1.09620	5.00002	13.95466	1.09620	0.25369	0.97066	-8.51295
4	1.09620	5.00000	13.95462	1.09620	0.25369	0.97066	-8.51293

```
p     =   9.0000000        9.0209898
a     =  -0.1411563       -0.1411331
e     =   8.0473056        8.0571895
i     = 180.0000000      180.0000000
Omega =   Undefined        Undefined
Argp  =   Undefined        Undefined
Nu    =  18.7455592       92.5176401
M     = 120.4376913      123.2503893

U     =   Undefined        Undefined
L     =   Undefined        Undefined
Cappi = 267.9536850      267.9633079
```

5. BMW Appendix D.3,4

Ro =	0.5000000	0.7000000	0.8000000	1.1747340
Vo =	0.0000000	0.1000000	0.9000000	0.9055385
Dt =	-20.00000 TU			
R =	0.0401556	0.2664818	1.9566242	1.9750958
V =	-0.2291452	-0.2755040	0.0410620	0.3606883
R =	-0.0345322	0.1776451	1.9665741	1.9748833
V =	-0.2269849	-0.2802384	0.0122192	0.3608393

Iterations:

#	X	tn	dt	xn	c	s	z
1	-4.27355	-3.33825	1.75082	-5.13271	0.07389	0.10191	16.11755
2	-5.13271	-4.97455	1.97359	-5.06579	0.05188	0.03832	23.24957
3	-5.06579	-4.84241	1.97510	-5.06583	0.05342	0.04210	22.64727
4	-5.06583	-4.84248	1.97510	-5.06583	0.05342	0.04210	22.64758

	Two-Body	Two-Body New	Perturbed New
p =	0.507500	0.5075000	0.5075000
a =	1.133000	1.1331277	1.1331277
e =	0.743051	0.7430509	0.7430509
i =	85.975300	85.9753157	85.9753157
Omega =	50.710600	50.7105931	51.1301413
Argp =	263.200600	263.2005657	266.1157703
Nu =	139.853500	180.0632823	180.4983038
M =	0.000000	180.2872918	182.2620600
U =	Undefined	Undefined	Undefined
L =	Undefined	Undefined	Undefined
Cappi =	Undefined	Undefined	Undefined

** Tests elliptical orbit with multi-revs, and backwards propogation **

6. BMW Appendix D.3,5

Ro =	0.0259170	-0.1506890	1.1388780	1.1490962
Vo =	0.0003610	0.0019740	0.0021770	0.0029608
Dt =	1.50000 TU			
R =	0.0085365	-0.0529898	0.3863987	0.3901086
V =	0.0412043	-0.2434089	1.8235400	1.8401749
R =	0.1795636	0.3559402	1.0177988	1.0930923
V =	-0.0502841	-0.0990878	-0.2771914	0.2986335

1	2.61073	1.37264	0.02563	3.10763	0.09159	0.16476	11.86306
2	3.10763	1.43116	0.24285	3.39110	0.07137	0.09370	16.80859
3	3.39110	1.52675	0.43715	3.32990	0.06081	0.06177	20.01489
4	3.32990	1.50137	0.39261	3.32642	0.06301	0.06808	19.29898
5	3.32642	1.50000	0.39010	3.32641	0.06314	0.06845	19.25860

	Two-Body	Two-Body New	Perturbed New
p =	0.000010	0.0000068	0.0000068
a =	0.575000	0.5745364	0.5745510
e =	0.999994	0.9999941	0.9999941
i =	87.675500	87.6755397	87.6755397
Omega =	262.160300	262.1602976	57.2818408
Argp =	262.713600	262.7135482	248.6861358
Nu =	179.999700	179.7256034	180.0445312
M =	179.668730	17.0114300	49.8512900
U =	Undefined	Undefined	Undefined
L =	Undefined	Undefined	Undefined
Cappi =	Undefined	Undefined	Undefined

** Tests rectilinear ellipse and x larger than TwoPi square root of a **

7. BMW Appendix D.3,6

```
Ro =    -0.5000000     0.0000000     0.0000000     0.5000000
Vo =     0.0000000     1.9990000     0.0000000     1.9990000
Dt = 1000.00000 TU

R  =   152.6766761    14.5709289     0.0000000   153.3703993
V  =     0.0950524     0.0025250     0.0000000     0.0950859
```

1	3.99900	12.60289	8.43757	15.70138	0.16613	0.49734	0.06395
2	15.70138	620.70914	113.73956	19.03611	0.15864	0.46025	0.98589
3	19.03611	1076.59133	160.51588	18.55895	0.15500	0.44246	1.44913
4	18.55895	1001.67034	153.52879	18.54807	0.15556	0.44518	1.37739
5	18.54807	1000.00086	153.37048	18.54807	0.15557	0.44524	1.37578
6	18.54807	1000.00000	153.37040	18.54807	0.15557	0.44524	1.37578

```
p     =   0.9990003       0.9990002
a     = 250.0625156     250.0625156
e     =   0.9980005       0.9980005
i     = 180.0000000     180.0000000
Omega =   Undefined       Undefined
Argp  =   Undefined       Undefined
Nu    =   0.0000000     174.5484021
M     =   0.0000000      14.4893779

U     =   Undefined       Undefined
L     =   Undefined       Undefined
Cappi = 180.0000000     180.0000000
```

** Tests x larger than TwoPi square root of a **

*8. Kaplan example problem pg. 307

```
Ro =     1.5679000     0.0000000     0.0000000     1.5679000
Vo =     0.0000000     1.1638000     0.0000000     1.1638000
Dt =    13.38600 TU

R  =    -4.8259941     7.3013686    -0.0000000     8.7521542
V  =    -0.4571857     0.3135849    -0.0000000     0.5543953
```

Iterations:

#	X	tn	dt	xn	c	s	z
1	-2.27833	-5.83264	4.58493	1.91337	0.17011	0.51729	-0.40924
2	1.91337	4.33083	3.67462	4.37762	0.16909	0.51214	-0.28863
3	4.37762	23.80414	13.75970	3.62047	0.17972	0.56621	-1.51083
4	3.62047	15.03431	9.58837	3.44857	0.17549	0.54457	-1.03341
5	3.44857	13.45554	8.78785	3.44065	0.17466	0.54031	-0.93760
6	3.44065	13.38614	8.75223	3.44064	0.17462	0.54012	-0.93330
7	3.44064	13.38600	8.75215	3.44064	0.17462	0.54012	-0.93329

```
p     =   3.3296105       3.3296116
a     = -12.6840963     -12.6840731
e     =   1.1236115       1.1236117
i     =   0.0000000       0.0000000
Omega =   Undefined       Undefined
Argp  =   Undefined       Undefined
Nu    =   0.0000000     123.4635459
M     =   0.0000000      16.9779486

U     =   Undefined       Undefined
L     =   Undefined       Undefined
Cappi =   0.0000000       0.0000076
```

**** Tests Hyperbolic first guess ****

KEPLER Test Cases (Continued)

*9. BMW problem pg. 224 #4.9

Ro =	0.0000000	1.1000000	0.0000000	1.1000000
Vo =	1.4142140	0.0000000	0.0000000	1.4142140
Dt =	2.22000 TU			

R =	2.4048811	0.0125729	0.0000000	2.4049140
V =	0.7747505	-0.6428154	-0.0000000	1.0067025

1	-2.92864	-8.65183	6.95066	-1.36450	0.18016	0.56845	-1.55945
2	-1.36450	-2.01771	2.24898	0.51978	0.16951	0.51427	-0.33852
3	0.51978	0.59992	1.26277	1.80274	0.16708	0.50205	-0.04912
4	1.80274	3.18987	3.14786	1.49464	0.17166	0.52511	-0.59089
5	1.49464	2.32559	2.48636	1.45217	0.17008	0.51715	-0.40617
6	1.45217	2.22171	2.40623	1.45146	0.16989	0.51618	-0.38342
7	1.45146	2.22000	2.40491	1.45146	0.16989	0.51617	-0.38305

p =	2.4200015	2.4200016
a =	-5.4999626	-5.4999621
e =	1.2000014	1.2000014
i =	180.0000000	180.0000000
Omega =	Undefined	Undefined
Argp =	Undefined	Undefined
Nu =	0.0000000	89.7004544
M =	0.0000000	9.8613428
U =	Undefined	Undefined
L =	Undefined	Undefined
Cappi =	270.0000000	270.0000017

*10. BMW problem pg. 225 #4.18

Ro =	0.2000000	0.0000000	0.0000000	0.2000000
Vo =	3.1622770	0.0000000	0.0000000	3.1622770
Dt =	219.60000 TU			

R =	60.1009021	0.0000000	0.0000000	60.1009021
V =	0.1824184	0.0000000	0.0000000	0.1824184

1	10.94471	258.57402	67.01540	10.36314	0.16667	0.50000	0.00000
2	10.36314	221.52504	60.45160	10.33130	0.16667	0.50000	0.00000
3	10.33130	219.60557	60.10196	10.33121	0.16667	0.50000	0.00000
4	10.33121	219.60000	60.10095	10.33121	0.16667	0.50000	0.00000

2.42000	-5.500	1.200001	180.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2.42000	-5.500	1.200001	180.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

KEPLER Test Cases (Continued)

11. A423 Skills Test #1

Ro =	1.6118775	2.2769723	-1.2822678	3.0703359
Vo =	-0.4250056	0.2604223	0.0542680	0.5013926
Dt =	214.17654 TU			
R =	1.8809513	-1.6342999	-0.0981493	2.4937015
V =	0.4770905	0.3231269	-0.2645644	0.6340510

p =	2.3437501	2.3437501
a =	2.5000000	2.5000000
e =	0.2500000	0.2500000
i =	24.9999999	24.9999999
Omega =	135.0000010	134.1680458
Argp =	79.9999986	81.4260368
Nu =	198.8108615	103.9177230
M =	209.9999997	75.0993655
U =	Undefined	Undefined
L =	Undefined	Undefined
Cappi =	Undefined	Undefined

12. A423 Skills Test #2

Ro =	2.7551415	-1.3550490	0.0000000	3.0703359
Vo =	0.1728093	0.4706713	0.0000000	0.5013926
Dt =	214.17654 TU			
R =	-1.2456159	-2.1633243	0.0000000	2.4963035
V =	0.4524628	-0.4432406	0.0000000	0.6333914

p =	2.3437501	2.3437501
a =	2.5000000	2.5000000
e =	0.2500000	0.2500000
i =	0.0000000	0.0000000
Omega =	Undefined	Undefined
Argp =	Undefined	Undefined
Nu =	198.8108623	104.1491546
M =	210.0000011	75.3374320
U =	Undefined	Undefined
L =	Undefined	Undefined
Cappi =	134.9999987	135.9180301

13. A423 Skills Test #3

Ro =	-2.3828847	0.1171153	0.7470906	2.5000000
Vo =	0.0296281	-0.6028275	0.1890006	0.6324555
Dt =	214.17654 TU			
R =	1.5826718	1.6215515	-1.0562767	2.5000000
V =	-0.4506182	0.4437419	0.0060304	0.6324555

p =	2.5000000	2.5000000
a =	2.5000000	2.5000000
e =	0.0000000	0.0000000
i =	25.0000000	25.0000000
Omega =	134.9999996	134.2688031
Argp =	Undefined	Undefined
Nu =	Undefined	Undefined
M =	45.0000003	271.2927914
U =	45.0000003	271.2927914
L =	Undefined	Undefined
Cappi =	Undefined	Undefined

14. A423 Skills Test #4

Ro =	-1.7677670	1.7677670	0.0000000	2.5000000
Vo =	-0.4472136	-0.4472136	0.0000000	0.6324555
Dt =	214.17654 TU			

R =	2.4995703	0.0463477	0.0000000	2.5000000
V =	-0.0117251	0.6323468	0.0000000	0.6324555

p =	2.5000000	2.5000000
a =	2.5000000	2.5000000
e =	0.0000000	0.0000000
i =	0.0000000	0.0000000
Omega =	Undefined	Undefined
Argp =	Undefined	Undefined
Nu =	Undefined	Undefined
M =	135.0000000	1.0622709

U =	Undefined	Undefined
L =	135.0000000	1.0622709
Cappi =	Undefined	Undefined

GAUSS Test Cases

*1. BMW Appendix D.4,1 and pg.275 5.11a

R1 =	0.5000000	0.6000000	0.7000000	1.0488088
R2 =	0.0000000	-1.0000000	0.0000000	1.0000000

Delta time = 20.0000000 TU Long Way

V12 =	-0.1229814	1.1921622	-0.1721740	1.2107927
V22 =	0.6698699	0.4804848	0.9378179	1.2486368

Iterations:

#	z	y	x	tn	vara	upper	lower
0	0.00000	2.99624	2.44795	1.28525	-0.670	39.47842	0.00000
1	19.73921	1.47495	4.79505	5.98349	-0.670	39.47842	19.73921
2	29.60881	1.18407	10.24734	40.59459	-0.670	29.60881	19.73921
3	24.67401	1.29916	6.54703	12.82559	-0.670	29.60881	24.67401
4	27.14141	1.23460	8.00517	21.34479	-0.670	27.14141	24.67401
5	25.90771	1.26506	7.20509	16.32057	-0.670	27.14141	25.90771
...							
20	26.85471	1.24136	7.80435	19.99996	-0.670	26.85475	26.85471
21	26.85473	1.24136	7.80436	20.00005	-0.670	26.85473	26.85471
22	26.85472	1.24136	7.80436	20.00001	-0.670	26.85472	26.85471

p	=	1.3282282	1.3282282
a	=	2.2680559	2.2680559
e	=	0.6437204	0.6437204
i	=	54.4623222	54.4623222
Omega	=	270.0000000	270.0000000
Argp	=	59.3433343	59.3433343
Nu	=	65.5518930	300.6566657
M	=	13.0845681	348.5688217
U	=	Undefined	Undefined
L	=	Undefined	Undefined
Cappi	=	Undefined	Undefined

2. BMW Appendix D.4,2

R1 =	0.3000000	0.7000000	0.4000000	0.8602325
R2 =	0.6000000	-1.4000000	0.8000000	1.7204651

Delta time = 5.0000000 TU Short Way

V12 =	0.7326124	-0.1048188	0.9768165	1.2255115
V22 =	-0.3438450	-0.1048188	-0.4584600	0.5825822

0	0.00000	1.16648	1.52741	1.67394	1.000	39.47842	0.00000
1	19.73921	3.43729	7.32002	26.03536	1.000	19.73921	0.00000
2	9.86960	2.58070	3.56865	6.21125	1.000	9.86960	0.00000
3	4.93480	1.95276	2.44978	3.30956	1.000	9.86960	4.93480
4	7.40220	2.28527	2.97388	4.53124	1.000	9.86960	7.40220
5	8.63590	2.43747	3.26097	5.30134	1.000	8.63590	7.40220
...							
17	8.17658	2.38187	3.15185	4.99985	1.000	8.17688	8.17658
18	8.17673	2.38189	3.15188	4.99994	1.000	8.17688	8.17673
19	8.17681	2.38190	3.15190	4.99999	1.000	8.17688	8.17681

p	=	0.8228737	0.8228737
a	=	1.2149570	1.2149570
e	=	0.5680790	0.5680790
i	=	126.8698976	126.8698976
Omega	=	90.0000000	90.0000000
Argp	=	301.1532226	301.1532226
Nu	=	94.3844552	203.3090996
M	=	31.1518847	245.0714534
U	=	Undefined	Undefined
L	=	Undefined	Undefined
Cappi	=	Undefined	Undefined

3. BMW Appendix D.4,3

R1 =	0.5000000	0.6000000	0.7000000	1.0488088
R2 =	0.0000000	1.0000000	0.0000000	1.0000000

Delta time = 1.2000000 TU Long Way

V12 =	-0.4053006	-0.9427690	-0.5674209	1.1726246
V22 =	0.2282041	1.1462875	0.3194858	1.2116614

0	0.00000	3.86474	2.78020	1.05725	-1.284	39.47842	0.00000
1	19.73921	0.94890	3.84604	2.25659	-1.284	19.73921	0.00000
2	9.86960	2.04881	3.17970	1.41934	-1.284	9.86960	0.00000
3	4.93480	2.85511	2.96220	1.21082	-1.284	4.93480	0.00000
4	2.46740	3.33287	2.86767	1.12883	-1.284	4.93480	2.46740
5	3.70110	3.08744	2.91396	1.16836	-1.284	4.93480	3.70110
...							
12	4.63602	2.91020	2.95033	1.20024	-1.284	4.63602	4.62638
13	4.63120	2.91109	2.95014	1.20008	-1.284	4.63120	4.62638
14	4.62879	2.91154	2.95004	1.19999	-1.284	4.63120	4.62879

p	=	0.1541483	0.1541483
a	=	1.8801350	1.8801350
e	=	0.9581295	0.9581295
i	=	125.5376778	125.5376778
Omega	=	90.0000000	90.0000000
Argp	=	208.0160964	208.0160964
Nu	=	207.0886763	151.9839036
M	=	346.1845632	12.8542076
U	=	Undefined	Undefined
L	=	Undefined	Undefined
Cappi	=	Undefined	Undefined

4. BMW Appendix D.4,4

R1 =	-0.2000000	0.6000000	0.3000000	0.7000000
R2 =	0.4000000	1.2000000	0.6000000	1.4000000

Delta time = 50.0000000 TU Short Way

V12 =	-0.1616701	1.4377416	0.7188708	1.6155536
V22 =	-0.1616701	-0.9613760	-0.4806880	1.0869415

Iterations:

#	z	y	x	tn	vara	upper	lower
0	0.00000	0.20263	0.63661	0.64694	1.342	39.47842	0.00000
1	19.73921	3.24923	7.11696	24.64273	1.342	39.47842	19.73921
2	29.60881	3.83177	18.43411	243.19086	1.342	29.60881	19.73921
3	24.67401	3.60129	10.90041	65.26346	1.342	24.67401	19.73921
4	22.20661	3.44164	8.74226	38.96151	1.342	24.67401	22.20661
5	23.44031	3.52541	9.73914	50.00039	1.342	23.44031	22.20661
...							
18	23.44016	3.52540	9.73901	49.99882	1.342	23.44031	23.44016
19	23.44024	3.52540	9.73907	49.99960	1.342	23.44031	23.44024
20	23.44027	3.52541	9.73910	50.00000	1.342	23.44031	23.44027

p	=	0.0453848	0.0453848
a	=	4.0464609	4.0464609
e	=	0.9943762	0.9943762
i	=	153.4349488	153.4349488
Omega	=	180.0000000	180.0000000
Argp	=	273.2706090	273.2706090
Nu	=	160.1278414	193.3309406
M	=	2.0933119	354.0421994
U	=	Undefined	Undefined
L	=	Undefined	Undefined
Cappi	=	Undefined	Undefined

5. BMW Appendix D.4,5 and pg. 275 5.11c

```
     R1 =      1.0000000      0.0000000      0.0000000      1.0000000
     R2 =      0.0000000      1.0000000      0.0000000      1.0000000
       Delta time =           0.0001000 TU Short Way

    V12 =      0.0000000      0.0000000      0.0000000      0.0000000
    V22 =      0.0000000      0.0000000      0.0000000      0.0000000

 0   0.00000   0.58579   1.08239   0.97672  1.000  0.00000-12.56637
 1  -2.86239   0.04886   0.27820   0.22517  1.000 -2.86239 -2.86239
 2  -2.86239   0.04886   0.27820   0.22517  1.000 -2.86239 -2.86239
 3  -2.86239   0.04886   0.27820   0.22517  1.000 -2.86239 -2.86239
 ...
28  -2.86239   0.04886   0.27820   0.22517  1.000 -2.86239 -2.86239
29  -2.86239   0.04886   0.27820   0.22517  1.000 -2.86239 -2.86239
30  -2.86239   0.04886   0.27820   0.22517  1.000 -2.86239 -2.86239

     p     =    Undefined         Undefined
     a     =    Undefined         Undefined
     e     =    Undefined         Undefined
     i     =    Undefined         Undefined
     Omega =    Undefined         Undefined
     Argp  =    Undefined         Undefined
     Nu    =    Undefined         Undefined
     M     =    Undefined         Undefined

     U     =    Undefined         Undefined
     L     =    Undefined         Undefined
     Cappi =    Undefined         Undefined
```

** Tests point where t is very close to 0. NOT possible to converge **

6. BMW Appendix D.4,6

```
     R1 =     -0.4000000      0.6000000     -1.2010000      1.4008572
     R2 =      0.2000000     -0.3000000      0.6000000      0.7000000
       Delta time =           5.0000000 TU Short Way

    V12 =      0.2551050     -0.3826576     -0.5738817      0.7354220
    V22 =     -0.7292157      1.0938236      0.4920219      1.4036706

 0   0.00000   2.10049   2.04963   1.43545  0.000 39.47842  0.00000
 1  19.73921   2.10108   5.72302  11.55672  0.000 19.73921  0.00000
 2   9.86960   2.10086   3.21983   3.38258  0.000 19.73921  9.86960
 3  14.80441   2.10098   4.20278   5.86034  0.000 14.80441  9.86960
 4  12.33701   2.10092   3.66271   4.39418  0.000 14.80441 12.33701
 5  13.57071   2.10095   3.91883   5.05635  0.000 13.57071 12.33701
 ...
17  13.47462   2.10095   3.89794   5.00013  0.000 13.47462 13.47432
18  13.47447   2.10095   3.89790   5.00004  0.000 13.47447 13.47432
19  13.47440   2.10095   3.89789   5.00000  0.000 13.47440 13.47432

     p     =   0.9334814        0.9334814
     a     =   1.1275842        1.1275842
     e     =   0.4148981        0.4148981
     i     =  90.0000000       90.0000000
     Omega = 303.6900675      303.6900675
     Argp  =  95.4911977       95.4911977
     Nu    = 143.5271501      323.5060831
     M     = 106.4464590      345.7059561

     U     =   Undefined        Undefined
     L     =   Undefined        Undefined
     Cappi =   Undefined        Undefined
```

** Tests two position vectors which are almost 180 deg apart **

*7. BMW Example problem pg. 236

```
   R1 =      0.5000000     0.6000000     0.7000000     1.0488088
   R2 =      0.0000000     1.0000000     0.0000000     1.0000000
     Delta time =         0.9668000 TU Long Way

  V12 =     -0.6309842    -1.1143338    -0.8833778     1.5557112
  V22 =      0.1785764     1.5552874     0.2500069     1.5853429
```

Iterations:

#	z	y	x	tn	vara	upper	lower
0	0.00000	3.86474	2.78020	1.05725	-1.284	0.00000	-12.56637
1	-6.28319	5.48830	2.58271	0.91188	-1.284	0.00000	-6.28319
2	-3.14159	4.62581	2.67724	0.97812	-1.284	-3.14159	-6.28319
3	-4.71239	5.04384	2.62895	0.94330	-1.284	-3.14159	-4.71239
4	-3.92699	4.83159	2.65284	0.96030	-1.284	-3.14159	-3.92699
...							
12	-3.63553	4.75448	2.66183	0.96682	-1.284	-3.63553	-3.63860
13	-3.63707	4.75489	2.66179	0.96678	-1.284	-3.63553	-3.63707
14	-3.63630	4.75468	2.66181	0.96680	-1.284	-3.63630	-3.63707

```
   p     =  0.0943930          0.0943930
   a     = -1.9481328         -1.9481328
   e     =  1.0239400          1.0239400
   i     = 125.5376778        125.5376778
   Omega =  90.0000000         90.0000000
   Argp  = 207.8180879        207.8180879
   Nu    = 207.2866848        152.1819121
   M     =  10.5153135          9.8480705

   U     =  Undefined          Undefined
   L     =  Undefined          Undefined
   Cappi =  Undefined          Undefined
```

** Tests Hyperbolic case with negative values of z **

*8. BMW Example problem pg. 236

```
   R1 =      0.5000000     0.6000000     0.7000000     1.0488088
   R2 =      0.0000000     1.0000000     0.0000000     1.0000000
     Delta time =         0.9668000 TU Short Way

  V12 =     -0.3616124     0.7697209    -0.5062574     0.9897123
  V22 =     -0.6018279    -0.0224179    -0.8425591     1.0356666
```

0	0.00000	0.23287	0.68246	0.67263	1.284	39.47842	0.00000
1	19.73921	3.14872	7.00602	23.47961	1.284	19.73921	0.00000
2	9.86960	2.04881	3.17970	5.09525	1.284	9.86960	0.00000
3	4.93480	1.24251	1.95412	2.40181	1.284	4.93480	0.00000
4	2.46740	0.76475	1.37366	1.50464	1.284	2.46740	0.00000
5	1.23370	0.50579	1.05940	1.09950	1.284	1.23370	0.00000
...							
16	0.83311	0.41872	0.94768	0.96696	1.284	0.83311	0.83251
17	0.83281	0.41866	0.94759	0.96686	1.284	0.83281	0.83251
18	0.83266	0.41862	0.94755	0.96681	1.284	0.83266	0.83251

```
   p     =  1.0721027          1.0721027
   a     =  1.0782895          1.0782895
   e     =  0.0757470          0.0757470
   i     =  54.4623222         54.4623222
   Omega = 270.0000000        270.0000000
   Argp  = 197.8449338        197.8449338
   Nu    = 287.0502935        342.1550662
   M     = 295.2054467        344.6774129

   U     =  Undefined          Undefined
   L     =  Undefined          Undefined
   Cappi =  Undefined          Undefined
```

9. BMW problem pg. 275 #5.11b

```
R1 =      1.2000000    0.0000000    0.0000000    1.2000000
R2 =      0.0000000    2.0000000    0.0000000    2.0000000
  Delta time =        10.0000000 TU Short Way

V12 =     0.7497686    0.7090867    0.0000000    1.0319675
V22 =    -0.4254520   -0.4661339   -0.0000000    0.6311024
```

0	0.00000	1.00911	1.42064	2.03409	1.549	39.47842	0.00000
1	19.73921	4.52702	8.40060	39.84524	1.549	19.73921	0.00000
2	9.86960	3.20000	3.97384	9.12942	1.549	19.73921	9.86960
3	14.80441	3.95748	5.76812	18.23108	1.549	14.80441	9.86960
4	12.33701	3.60388	4.79715	12.81244	1.549	12.33701	9.86960
5	11.10330	3.40844	4.36982	10.80249	1.549	11.10330	9.86960
...							
19	10.53909	3.31475	4.18513	10.00008	1.549	10.53909	10.53901
20	10.53905	3.31475	4.18512	10.00003	1.549	10.53905	10.53901
21	10.53903	3.31474	4.18511	10.00000	1.549	10.53903	10.53901

```
p     =  0.7240377       0.7240377
a     =  1.6619309       1.6619309
e     =  0.7512253       0.7512253
i     =  0.0000000       0.0000000
Omega =  Undefined       Undefined
Argp  =  Undefined       Undefined
Nu    = 121.8693704     211.8693704
M     =  28.2972591     295.7231038

U     =  Undefined       Undefined
L     =  Undefined       Undefined
Cappi = 238.1306296     238.1306296
```

*10. BMW problem pg. 275 #5.11d

```
R1 =      4.0000000    0.0000000    0.0000000    4.0000000
R2 =     -2.0000000    0.0000000    0.0000000    2.0000000
  Delta time =        10.0000000 TU Short Way

V12 =     0.0000000    0.0000000    0.0000000    0.0000000
V22 =     0.0000000    0.0000000    0.0000000    0.0000000
```

Iterations:

#	z	y	x	tn	vara	upper	lower

None.

```
p     =  Undefined       Undefined
a     =  Undefined       Undefined
e     =  Undefined       Undefined
i     =  Undefined       Undefined
Omega =  Undefined       Undefined
Argp  =  Undefined       Undefined
Nu    =  Undefined       Undefined
M     =  Undefined       Undefined

U     =  Undefined       Undefined
L     =  Undefined       Undefined
Cappi =  Undefined       Undefined
```

** NOT POSSIBLE since the vectors are Co-linear, i.e. Nu = Pi **

11. BMW problem pg. 275 #5.11e

```
   R1 =      2.0000000     0.0000000     0.0000000      2.0000000
   R2 =     -2.0000000    -0.2000000     0.0000000      2.0099751
     Delta time =         20.0000000 TU Long Way

  V12 =      0.3083363     0.7157383    -0.0000000      0.7793283
  V22 =      0.3778475    -0.6779535     0.0000000      0.7761377


 0   0.00000    4.20973    2.90163    3.78189 -0.141 39.47842  0.00000
 1  19.73921    3.88899    7.78614   28.82271 -0.141 19.73921  0.00000
 2   9.86960    4.00998    4.44842    8.63618 -0.141 19.73921  9.86960
 3  14.80441    3.94091    5.75603   14.77378 -0.141 19.73921 14.80441
 4  17.27181    3.91295    6.65020   20.22782 -0.141 17.27181 14.80441
...
20  17.18766    3.91384    6.61618   20.00007 -0.141 17.18766 17.18762
21  17.18764    3.91384    6.61617   20.00002 -0.141 17.18764 17.18762
22  17.18763    3.91384    6.61617   19.99999 -0.141 17.18764 17.18763

   p      =   2.0491252          2.0491252
   a      =   2.5468139          2.5468139
   e      =   0.4420591          0.4420591
   i      =   0.0000000          0.0000000
   Omega =   Undefined          Undefined
   Argp  =   Undefined          Undefined
   Nu    =  86.8147745        272.5253677
   M     =  38.8020471        320.7420307

   U     =   Undefined          Undefined
   L     =   Undefined          Undefined
   Cappi = 273.1852255        273.1852255
```

*12. BMW problem pg. 274 #5.8

```
   R1 =      1.0000000     0.0000000     0.0000000      1.0000000
   R2 =      1.0000000     1.0000000     1.0000000      1.7320508
     Delta time =          1.0922000 TU Short Way

  V12 =      0.3628742     1.0086839     1.0086839      1.4719253
  V22 =     -0.2095055     0.7991784     0.7991784      1.1494628

 0   0.00000    0.39451    0.88827    1.15499  1.653  0.00000-12.56637
 1  -1.12642    0.05758    0.32393    0.40261  1.653  0.00000 -1.12642
 2  -0.56321    0.22800    0.65969    0.83846  1.653  0.00000 -0.56321
 3  -0.28161    0.31174    0.78042    1.00322  1.653  0.00000 -0.28161
 4  -0.14080    0.35325    0.83562    1.08032  1.653  0.00000 -0.14080
 5  -0.07040    0.37391    0.86223    1.11792  1.653 -0.07040 -0.14080
...
12  -0.11625    0.35987    0.84421    1.09243  1.653 -0.11825 -0.11880
13  -0.11853    0.35979    0.84411    1.09228  1.653 -0.11853 -0.11880
14  -0.11866    0.35975    0.84405    1.09221  1.653 -0.11866 -0.11880

   p      =   2.0348865          2.0348865
   a      =  -6.0036903         -6.0036903
   e      =   1.1571254          1.1571254
   i      =  45.0000000         45.0000000
   Omega =   0.0000000          0.0000000
   Argp  = 333.4263200        333.4263200
   Nu    =  26.5736800         81.3092903
   M     =   1.1720949          5.4261265

   U     =   Undefined          Undefined
   L     =   Undefined          Undefined
   Cappi =   Undefined          Undefined
```

** Tests Hyperbola and lower bounds on a negative iteration **

*13. BMW problem pg. 274 #5.10

R1 =	1.0000000	0.0000000	0.0000000	1.0000000
R2 =	1.0000000	0.1250000	0.1250000	1.0155048
Delta time =		0.1250000 TU Short Way		
V12 =	0.0618607	1.0025629	1.0025629	1.4191869
V22 =	-0.0609162	0.9949484	0.9949484	1.4083875

Iterations:

#	z	y	x	tn	vara	upper	lower
0	0.00000	0.00777	0.12464	0.12544	1.420	0.00000	-12.56637
1	-0.03070	0.00006	0.01071	0.01076	1.420	0.00000	-0.03070
2	-0.01535	0.00391	0.08842	0.08893	1.420	0.00000	-0.01535
3	-0.00768	0.00584	0.10805	0.10871	1.420	0.00000	-0.00768
4	-0.00384	0.00680	0.11664	0.11737	1.420	0.00000	-0.00384
5	-0.00192	0.00729	0.12070	0.12147	1.420	0.00000	-0.00192
...							
11	-0.00021	0.00771	0.12421	0.12502	1.420	-0.00021	-0.00024
12	-0.00022	0.00771	0.12418	0.12498	1.420	-0.00021	-0.00022
13	-0.00022	0.00771	0.12420	0.12500	1.420	-0.00021	-0.00022

p	=	2.0102648	2.0102648
a	=	-70.9646068	-70.9646068
e	=	1.0140649	1.0140649
i	=	45.0000000	45.0000000
Omega	=	0.0000000	0.0000000
Argp	=	355.0381799	355.0381799
Nu	=	4.9618201	14.9868079
M	=	0.0058393	0.0178196
U	=	Undefined	Undefined
L	=	Undefined	Undefined
Cappi	=	Undefined	Undefined

** Tests first guess too close and solution near zero **

14. A423 test case

R1 =	1.0500000	0.0000000	0.0000000	1.0500000
R2 =	-3.2500000	2.6037000	0.0000000	4.1643431
Delta time =		2.0000000 TU Short Way		
V12 =	-1.7964252	1.9359850	0.0000000	2.6410569
V22 =	-2.1040017	1.0601246	0.0000000	2.3559897

Iterations:

#	z	y	x	tn	vara	upper	lower
0	0.00000	3.82866	2.76719	5.44876	0.980	0.00000	-12.56637
1	-6.28319	2.58977	1.77414	2.84747	0.980	-6.28319	-12.56637
2	-9.42478	1.84751	1.33487	1.96246	0.980	-6.28319	-9.42478
3	-7.85398	2.22959	1.55259	2.38257	0.980	-7.85398	-9.42478
4	-8.63938	2.04134	1.44353	2.16772	0.980	-8.63938	-9.42478
5	-9.03208	1.94513	1.38919	2.06399	0.980	-9.03208	-9.42478
...							
13	-9.27905	1.88390	1.35503	1.99989	0.980	-9.27752	-9.27905
14	-9.27828	1.88409	1.35514	2.00009	0.980	-9.27828	-9.27905
15	-9.27867	1.88399	1.35508	1.99999	0.980	-9.27828	-9.27867

p	=	4.1322119	4.1322119
a	=	-0.1972223	-0.1972223
e	=	4.6853013	4.6853013
i	=	0.0000000	0.0000000
Omega	=	Undefined	Undefined
Argp	=	Undefined	Undefined
Nu	=	308.7939154	90.0943554
M	=	196.7341763	30.3576180
U	=	Undefined	Undefined
L	=	Undefined	Undefined
Cappi	=	51.2060846	51.2060846

15. A423 Test Case

```
   R1 =      1.0500000      0.0000000      0.0000000      1.0500000
   R2 =      0.0000000      0.9000000      0.0000000      0.9000000
     Delta time =          35.0000000 TU Short Way

  V12 =      1.1418714      0.5381541      0.0000000      1.2623312
  V22 =     -0.6278465     -1.2315637     -0.0000000      1.3823677
```

Iterations:

#	z	y	x	tn	vara	upper	lower
0	0.00000	0.57523	1.07259	0.94295	0.972	39.47842	0.00000
1	19.73921	2.78270	6.58624	19.23557	0.972	39.47842	19.73921
2	29.60881	3.20479	16.85862	185.74593	0.972	29.60881	19.73921
3	24.67401	3.03779	10.01135	50.28323	0.972	24.67401	19.73921
4	22.20661	2.92211	8.05544	30.19576	0.972	24.67401	22.20661
5	23.44031	2.98281	8.95836	38.63162	0.972	23.44031	22.20661
...							
22	22.95547	2.95965	8.58749	35.00001	0.972	22.95547	22.95546
23	22.95546	2.95965	8.58749	34.99998	0.972	22.95547	22.95546
24	22.95547	2.95965	8.58749	34.99999	0.972	22.95547	22.95547

```
   p     =   0.3192949          0.3192949
   a     =   3.2125232          3.2125232
   e     =   0.9490044          0.9490044
   i     =   0.0000000          0.0000000
   Omega =   Undefined          Undefined
   Argp  =   Undefined          Undefined
   Nu    = 137.1641835        227.1641835
   M     =   6.4927409        354.7668629

   U     =   Undefined          Undefined
   L     =   Undefined          Undefined
   Cappi = 222.8358165        222.8358165
```

16. A423 Test Case

```
   R1 =      1.0500000      0.0000000      0.0000000      1.0500000
   R2 =     -3.2500000      2.6037000      0.0000000      4.1643431
     Delta time =          10.0000000 TU Short Way

  V12 =      0.2035271      1.2213287      0.0000000      1.2381709
  V22 =     -0.2840267     -0.1670384     -0.0000000      0.3295042
```

Iterations:

#	z	y	x	tn	vara	upper	lower
0	0.00000	3.82866	2.76719	5.44876	0.980	39.47842	0.00000
1	19.73921	6.05365	9.71433	58.92834	0.980	19.73921	0.00000
2	9.86960	5.21434	5.07265	15.46269	0.980	9.86960	0.00000
3	4.93480	4.59908	3.75957	9.01247	0.980	9.86960	4.93480
4	7.40220	4.92488	4.36568	11.72713	0.980	7.40220	4.93480
5	6.16850	4.76667	4.05153	10.26622	0.980	6.16850	4.93480
...							
19	5.92160	4.73389	3.99141	9.99998	0.980	5.92167	5.92160
20	5.92163	4.73389	3.99142	10.00002	0.980	5.92163	5.92160
21	5.92162	4.73389	3.99141	10.00000	0.980	5.92162	5.92160

```
   p     =   1.6445373          1.6445373
   a     =   2.6903791          2.6903791
   e     =   0.6234854          0.6234854
   i     =   0.0000000          0.0000000
   Omega =   Undefined          Undefined
   Argp  =   Undefined          Undefined
   Nu    =  24.7473677        166.0478077
   M     =   4.5968877        134.4351016

   U     =   Undefined          Undefined
   L     =   Undefined          Undefined
   Cappi = 335.2526323        335.2526323
```

PKEPLER Test Cases

1. A423 test case

Initial Target location :

Ro =	-3.25000000	2.60370000	0.00000000	4.16434313
Vo =	-0.30640000	-0.38240000	0.00000000	0.49001094

Dt = 10.0000000000000 TU

Final Target location :

R2 =	-3.65171978	-2.00172677	0.00000000	4.16436879
V2 =	0.23556336	-0.42967158	0.00000000	0.49000792

p =	4.16394100	4.16394100
a =	4.16394100	4.16394100
e =	0.00011993	0.00011993
i =	0.00000000	0.00000000
Omega =	***************	***************
Argp =	***************	***************
Nuo =	143.63583857	211.05887378
M =	143.62768933	211.06596478
U =	***************	***************
L =	***************	***************
CapPi =	357.66460145	357.67091784

----------- GAUSS -----------

Interceptor Position

R1 =	1.05000000	0.00000000	0.00000000	1.05000000

Final Target Position

R2 =	-3.65171978	-2.00172677	0.00000000	4.16436879

Delta time = 10.0000000000000 Short Way

Iterations :

	z	y	x	tn	vara	Upper z	Lower z
0	0.00000	4.17679	2.89026	5.52343	0.73368	39.47842	0.00000
1	19.73921	5.84283	9.54368	55.36444	0.73368	19.73921	0.00000
2	9.86960	5.21437	5.07266	14.90071	0.73368	9.86960	0.00000
3	4.93480	4.75367	3.82223	8.86219	0.73368	9.86960	4.93480
4	7.40220	4.99762	4.39781	11.40528	0.73368	7.40220	4.93480
5	6.16850	4.87916	4.09906	10.03697	0.73368	6.16850	4.93480
6	5.55165	4.81731	3.95806	9.42764	0.73368	6.16850	5.55165
7	5.86008	4.84845	4.02790	9.72656	0.73368	6.16850	5.86008
8	6.01429	4.86386	4.06331	9.88030	0.73368	6.16850	6.01429
9	6.09140	4.87152	4.08114	9.95826	0.73368	6.16850	6.09140
10	6.12995	4.87534	4.09009	9.99752	0.73368	6.16850	6.12995
11	6.14923	4.87725	4.09457	10.01722	0.73368	6.14923	6.12995
12	6.13959	4.87630	4.09233	10.00737	0.73368	6.13959	6.12995
13	6.13477	4.87582	4.09121	10.00244	0.73368	6.13477	6.12995
14	6.13236	4.87558	4.09065	9.99998	0.73368	6.13477	6.13236
15	6.13356	4.87570	4.09093	10.00121	0.73368	6.13356	6.13236
16	6.13296	4.87564	4.09079	10.00060	0.73368	6.13296	6.13236
17	6.13266	4.87561	4.09072	10.00029	0.73368	6.13266	6.13236
18	6.13251	4.87560	4.09069	10.00014	0.73368	6.13251	6.13236
19	6.13243	4.87559	4.09067	10.00006	0.73368	6.13243	6.13236
20	6.13240	4.87559	4.09066	10.00002	0.73368	6.13240	6.13236
21	6.13238	4.87558	4.09065	10.00000	0.73368	6.13238	6.13236

Transfer orbit velocities :

V1t =	0.10732319	-1.23562515	0.00000000	1.24027730
V2t =	-0.26316944	0.21102745	0.00000000	0.33732883

p =	2.72870561	2.72870561
a =	2.72870561	2.72870561
e =	0.61897296	0.61897296
i =	180.00000000	180.00000000
Omega =	***************	***************
Argp =	***************	***************
Nuo =	13.00030198	164.27051036
M =	2.41894132	129.53127945
U =	***************	***************
L =	***************	***************
CapPi =	346.99969802	346.99969802

If the velocity of the interceptor is

V1 =	0.00000000	0.97590000	0.00000000	0.97590000

DeltaV1 = V1t - V1 =	0.10732319	-2.21152515	0.00000000	2.21412776
DeltaV2 = V2 - V2t =	0.49873280	-0.64069902	0.00000000	0.81192958

PKEPLER Test Cases (Continued)

2. A423 test case

Initial Target location :

Ro =	-3.25000000	2.60370000	0.00000000	4.16434313
Vo =	-0.30640000	-0.38240000	0.00000000	0.49001094

Dt = 2.00000000000000 TU

Final Target location :

R2 =	-3.76764653	1.77400005	0.00000000	4.16440110
V2 =	-0.20875833	-0.44331027	0.00000000	0.49000412

p =	4.16394100	4.16394100
a =	4.16394100	4.16394100
e =	0.00011993	0.00011993
i =	0.00000000	0.00000000
Omega =	**************	**************
Argp =	**************	**************
Nuo =	143.63583857	157.12068816
M =	143.62768933	157.11534442
U =	**************	**************
L =	**************	**************
CapPi =	357.66460145	357.66586473

----------- GAUSS -----------

Interceptor Position

R1 =	1.05000000	0.00000000	0.00000000	1.05000000

Final Target Position

R2 =	-3.76764653	1.77400005	0.00000000	4.16440110

Delta time = 2.00000000000000 Short Way

Iterations :

	z	y	x	tn	vara	Upper z	Lower z
0	0.00000	4.30161	2.93313	5.54438	0.64544	0.00000	-12.56637
1	-6.28319	3.48552	2.05822	3.18899	0.64544	-6.28319	-12.56637
2	-9.42478	2.99657	1.70004	2.42000	0.64544	-9.42478	-12.56637
3	-10.99557	2.72987	1.53469	2.10002	0.64544	-10.99557	-12.56637
4	-11.78097	2.59072	1.45474	1.95288	0.64544	-10.99557	-11.78097
5	-11.38827	2.66079	1.49451	2.02546	0.64544	-11.38827	-11.78097
6	-11.58462	2.62588	1.47458	1.98893	0.64544	-11.38827	-11.58462
7	-11.48645	2.64336	1.48453	2.00714	0.64544	-11.48645	-11.58462
8	-11.53554	2.63463	1.47955	1.99802	0.64544	-11.48645	-11.53554
9	-11.51099	2.63900	1.48204	2.00257	0.64544	-11.51099	-11.53554
10	-11.52326	2.63681	1.48080	2.00029	0.64544	-11.52326	-11.53554
11	-11.52940	2.63572	1.48017	1.99916	0.64544	-11.52326	-11.52940
12	-11.52633	2.63627	1.48049	1.99973	0.64544	-11.52326	-11.52633
13	-11.52480	2.63654	1.48064	2.00001	0.64544	-11.52480	-11.52633
14	-11.52556	2.63640	1.48056	1.99987	0.64544	-11.52480	-11.52556
15	-11.52518	2.63647	1.48060	1.99994	0.64544	-11.52480	-11.52518
16	-11.52499	2.63651	1.48062	1.99997	0.64544	-11.52480	-11.52499
17	-11.52489	2.63652	1.48063	1.99999	0.64544	-11.52480	-11.52489

Transfer orbit velocities :

V1t =	-2.08117693	1.69270886	0.00000000	2.68264062
V2t =	-2.32085567	0.62103856	0.00000000	2.40251117

p =	-0.18897166	-0.18897166
a =	-0.18897166	-0.18897166
e =	4.20910234	4.20910234
i =	0.00000000	0.00000000
Omega =	**************	**************
Argp =	**************	**************
Nuo =	298.50180289	93.28835579
M =	230.01866192	1161.08942741
U =	**************	**************
L =	**************	**************
CapPi =	61.49819711	61.49819711

If the velocity of the interceptor is

V1 =	0.00000000	0.97590000	0.00000000	0.97590000

DeltaV1 = V1t - V1 =	-2.08117693	0.71680886	0.00000000	2.20116159
DeltaV2 = V2 - V2t =	2.11209734	-1.06434883	0.00000000	2.36512021

PKEPLER Test Cases (Continued)

3. A423 test case

Initial Target location :

Ro =	-3.25000000	2.60370000	0.00000000	4.16434313
Vo =	-0.30640000	-0.38240000	0.00000000	0.49001094

Dt = 35.0000000000000 TU

Final Target location :

R2 =	3.97397194	1.24178724	0.00000000	4.16347073
V2 =	-0.14616128	0.46781218	0.00000000	0.49011361

p =	4.16394100	4.16394100
a =	4.16394100	4.16394100
e =	0.00011993	0.00011993
i =	0.00000000	0.00000000
Omega =	**************	**************
Argp =	**************	**************
Nuo =	143.63583857	19.66627822
M =	143.62768933	19.66165341
U =	**************	**************
L =	**************	**************
CapPi =	357.66460145	357.68670880

----------- GAUSS -----------

Interceptor Position

R1 =	1.05000000	0.00000000	0.00000000	1.05000000

Final Target Position

R2 =	3.97397194	1.24178724	0.00000000	4.16347073

Delta time = 35.0000000000000 Short Way

Iterations :

	z	y	x	tn	vara	Upper z	Lower z
0	0.00000	1.07963	1.46944	3.56604	2.92307	39.47842	0.00000
1	19.73921	7.71734	10.96827	89.47046	2.92307	19.73921	0.00000
2	9.86960	5.21347	5.07222	19.89619	2.92307	19.73921	9.86960
3	14.80441	6.64271	7.47304	40.47799	2.92307	14.80441	9.86960
4	12.33701	5.97553	6.17712	28.22152	2.92307	14.80441	12.33701
5	13.57071	6.32058	6.79714	33.73129	2.92307	14.80441	13.57071
6	14.18756	6.48446	7.12753	36.92987	2.92307	14.18756	13.57071
7	13.87913	6.40323	6.96052	35.28964	2.92307	13.87913	13.57071
8	13.72492	6.36208	6.87839	34.50055	2.92307	13.87913	13.72492
9	13.80202	6.38270	6.91934	34.89258	2.92307	13.87913	13.80202
10	13.84058	6.39298	6.93990	35.09048	2.92307	13.84058	13.80202
11	13.82130	6.38784	6.92962	34.99137	2.92307	13.84058	13.82130
12	13.83094	6.39041	6.93476	35.04088	2.92307	13.83094	13.82130
13	13.82612	6.38913	6.93219	35.01612	2.92307	13.82612	13.82130
14	13.82371	6.38848	6.93090	35.00374	2.92307	13.82371	13.82130
15	13.82251	6.38816	6.93026	34.99756	2.92307	13.82371	13.82251
16	13.82311	6.38832	6.93058	35.00065	2.92307	13.82311	13.82251
17	13.82281	6.38824	6.93042	34.99910	2.92307	13.82311	13.82281
18	13.82296	6.38828	6.93050	34.99988	2.92307	13.82311	13.82296
19	13.82303	6.38830	6.93054	35.00026	2.92307	13.82303	13.82296
20	13.82300	6.38829	6.93052	35.00007	2.92307	13.82300	13.82296
21	13.82298	6.38829	6.93051	34.99997	2.92307	13.82300	13.82298
22	13.82299	6.38829	6.93052	35.00002	2.92307	13.82299	13.82298
23	13.82298	6.38829	6.93051	35.00000	2.92307	13.82299	13.82298

Transfer orbit velocities :

V1t =	1.26044599	0.16808010	0.00000000	1.27160333
V2t =	-0.42955162	-0.08981634	0.00000000	0.43884117

p =	3.47479351	3.47479351
a =	3.47479351	3.47479351
e =	0.99550811	0.99550811
i =	0.00000000	0.00000000
Omega =	**************	**************
Argp =	**************	**************
Nuo =	167.08810965	184.44109668
M =	4.81579179	314.41306557
U =	**************	**************
L =	**************	**************
CapPi =	192.91189035	192.91189035

If the velocity of the interceptor is

V1 =	0.00000000	0.97590000	0.00000000	0.97590000

DeltaV1 = V1t - V1 =	1.26044599	-0.80781990	0.00000000	1.49709622
DeltaV2 = V2 - V2t =	0.28339034	0.55762852	0.00000000	0.62550751

IJKtoLatLon:

	i	j	k		
r =	1.125	0.784	1.372 DU	JD =	2446066.7835

Geocentric Latitude = 44.923
Longitude = -168.059

SUN

Date	Time	Rt Asc	Declination	x	y	z	Magnitude
7 Mar 1960	0: 0: 0.00	347.5098316	-5.3581087	0.9647718	-0.2137111	-0.0926796	0.9924951
1 Jan 1987	0: 0: 0.00	280.9058053	-23.0621533	0.1711689	-0.8883825	-0.3851906	0.9833077
*				0.1742717	-0.8878979	-0.3849824	0.9833335
14 May 1987	0: 0: 0.00	50.2799618	18.4437210	0.6126212	0.7373809	0.3197175	1.0105714
*		50.2831665	18.4464166	0.6099669	0.7391757	0.3204954	1.0105234
7 Dec 1987	0: 0: 0.00	253.0019878	-22.5209713	-0.2660578	-0.8703439	-0.3773668	0.9852365
*				-0.2632428	-0.8710383	-0.3776657	0.9852085
1 Jan 1988	0: 0: 0.00	280.6376087	-23.0804049	0.1669865	-0.8890567	-0.3854802	0.9833114
*				0.1698618	-0.8885706	-0.3852661	
7 Jan 1989	0: 0: 0.00	288.0699572	-22.4012346	0.2819902	-0.8642844	-0.3747367	0.9833278
*		288.0718324	-22.4032777	0.2846084	-0.8635956	-0.3744404	
7 Sep 1989	0: 0: 0.00	165.5883823	6.1590506	-0.9703890	0.2493633	0.1081185	1.0077333
*		165.5914994	6.1581388	-0.9711011	0.2470317	0.1071008	
27 Oct 1989	0: 0: 0.00	211.3147039	-12.6993087	-0.8282793	-0.5038929	-0.2184768	0.9938244
*		211.3141660	-12.7006667	-0.8269128	-0.5058353	-0.2193263	

MOON

Date	Time	Rt Asc	Declination	x	y	z	Magnitude
7 Mar 1960	0: 0: 0.00	93.2827108	18.1740408	-3.4450973	60.0642320	19.7498143	63.3200725
1 Jan 1987	0: 0: 0.00	295.0571035	-26.4619655	21.3030552	-45.5659292	-25.0362158	56.1849453
14 May 1987	0: 0: 0.00	235.9616350	-23.6405903	-29.4768585	-43.6381820	-23.0509295	57.4838850
		235.8341659	-23.6722222				
7 Dec 1987	0: 0: 0.00	93.8507651	28.3624008	-3.6947803	54.8920738	29.6995740	62.5192520
		93.8729164	28.3913889				
1 Jan 1988	0: 0: 0.00	62.0701366	25.7416698	26.0736204	49.1824943	26.8396283	61.7977078
1 Jan 1989	0: 0: 0.00	196.2814860	-10.6881272	-59.4985477	-17.3777252	-11.6987324	63.0785260
		196.3591660	-10.7161111				
7 Sep 1989	0: 0: 0.00	235.0997646	-24.9840899	-32.7168135	-46.8980417	-26.6446131	63.0840945
		235.0529159	-24.9547222				
27 Oct 1989	0: 0: 0.00	183.1465367	-4.9906999	-63.1781196	-3.4730719	-5.5253644	63.5142858
		183.1216661	-4.9911111				

* Astronomical Almanac Values

SUNRiseSet

JDate	Day Mon Yr	Lat	SunRise hr	SunRise min	Sunset hr	Sunset min
2448258.5	2 Jan 91	40.0	7	22	16	46
2448430.5	23 Jun 91	40.0	4	32	19	33
2448430.5	23 Jun 91	64.0	1	31	22	33
2448438.5	1 Jul 91	-55.0	8	26	15	41
2448614.5	24 Dec 91	40.0	7	19	16	39

MISC ICBM Test Cases

PATH and RNGAZ Test Cases

0.0000000	0.0000000	1000.0000000	169.8570000	-8.8416236	1.5946625
0.0000000	0.0000000	0.0000000	169.8570000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	10.0000000	0.0000000	0.0898315	0.0000000
0.0000000	0.0000000	10.0000000	90.0000000	-0.0000000	0.0898315
0.0000000	0.0000000	10.0000000	180.0000000	-0.0898315	0.0000000
0.0000000	0.0000000	10.0000000	270.0000000	0.0000000	359.9101685
0.0000000	0.0000000	1536.8540000	270.0000000	0.0000000	346.1942054
0.0000000	0.0000000	6370.0000000	270.0000000	0.0000000	302.7773155
0.0000000	0.0000000	6370.0000000	180.0000000	-57.2226845	0.0000000
0.0000000	0.0000000	6370.0000000	90.0000000	-0.0000000	57.2226845
0.0000000	0.0000000	6370.0000000	0.0000000	57.2226845	0.0000000
0.0000000	0.0000000	8000.0000000	0.0000000	71.8652238	0.0000000
0.0000000	0.0000000	8000.0000000	90.0000000	-0.0000000	71.8652238
0.0000000	0.0000000	8000.0000000	180.0000000	-71.8652238	360.0000000
0.0000000	0.0000000	8000.0000000	270.0000000	0.0000000	288.1347762
-20.0000000	0.0000000	8000.0000000	270.0000000	-6.1109959	287.1068112
-20.0000000	0.0000000	6000.0000000	270.0000000	-11.6259897	304.4213918
-20.0000000	0.0000000	6000.0000000	180.0000000	-73.8989179	0.0000000
-20.0000000	0.0000000	6000.0000000	90.0000000	-11.6259897	55.5786082
-20.0000000	0.0000000	6000.0000000	90.0000000	-11.6259897	55.5786082
-20.0000000	0.0000000	6000.0000000	0.0000000	33.8989179	0.0000000
30.0000000	0.0000000	6000.0000000	0.0000000	83.8989179	0.0000000
30.0000000	0.0000000	6000.0000000	90.0000000	17.1339014	57.7258837
30.0000000	0.0000000	6000.0000000	180.0000000	-23.8989179	0.0000000
30.0000000	0.0000000	6000.0000000	270.0000000	17.1339014	302.2741163
86.0000000	0.0000000	6000.0000000	270.0000000	35.9993679	272.9120869
86.0000000	0.0000000	6000.0000000	180.0000000	32.1010821	0.0000000
86.0000000	0.0000000	6000.0000000	270.0000000	35.9993679	272.9120869
86.0000000	0.0000000	6000.0000000	90.0000000	35.9993679	87.0879131
86.0000000	-100.0000000	6000.0000000	90.0000000	35.9993679	347.0879131
86.0000000	-100.0000000	6000.0000000	0.0000000	40.1010821	80.0000000
86.0000000	-100.0000000	6000.0000000	180.0000000	32.1010821	260.0000000
86.0000000	-100.0000000	6000.0000000	270.0000000	35.9993679	172.9120869
86.0000000	140.0000000	6000.0000000	270.0000000	35.9993679	52.9120869
86.0000000	140.0000000	6000.0000000	180.0000000	32.1010821	140.0000000
86.0000000	140.0000000	6000.0000000	90.0000000	35.9993679	227.0879131
86.0000000	140.0000000	6000.0000000	0.0000000	40.1010821	320.0000000

GEOCENTRIC and INVGEOCENTRIC Test Cases

Geocentric Latitude Deg	Geodetic Latitude Deg
-20.0000	0.0000
0.0000	
10.0000	
20.0000	
50.0000	
70.0000	
90.0000	

TRAJEC Test Cases

Alt of burnout 0.0 ft

Latitudes	Longitudes	Q	Type Trajectory
47.3000000	-111.1600000	0.870000	High
43.3000000	61.0000000		
89.3649491	9948.0604758		Range in deg and km
34.6748938			Flight Path Angle deg
42.7266242			Time of Flight min
357.9239768			Azimuth deg
-136.2692682	4.2029324	2.4294622	Influence Coefficients
7.3897206			Needed velocity km/s
47.3000000	-111.1600000	0.870000	Low
43.3000000	61.0000000		
89.3907074	9950.9278763		Range in deg and km
10.6511053			Flight Path Angle deg
25.4576416			Time of Flight min
1.0690019			Azimuth deg
350.6706321	10.8307851	6.2606248	Influence Coefficients
7.3745838			Needed velocity km/s

PREDICT TEST Cases

Epoch Data :			Site Data :		Time Data :
2447776.500			76.570	Latitude	2447826.50
12.53536954	n	rev/day	-68.270	Longitude	2900.00 min
0.00243456	ndot	rev/2day2	2519.685	Altitude ft	2.00 min
0.1604069	e				
-0.000232	edot	/day			
74.4536	i				
325.0201	omega	deg			
-1.8465	omega dot	deg/day			
324.6788	argp	deg			
-4.5783	argp dot	deg/day			
36.84884	M	deg			

NOTE!! These results are produced using the Dot terms above. If the user calculates their own terms, secular, J2 only, the values will differ substantially.

Results:	Range km	Az	El	JD		Universal Hr Min Sec
Eye	1859.5720	267.10873	0.30459	2447826.5000000	27 Oct 1989	0: 0: 0.00
Eye	926.2845	259.58111	10.93659	2447826.5013889	27 Oct 1989	0: 1:60.00
Radar Nite	339.4721	161.27447	39.40792	2447826.5027778	27 Oct 1989	0: 3:60.00
Radar Nite	1121.4410	107.98459	6.52213	2447826.5041667	27 Oct 1989	0: 5:60.00
Radar Nite	1102.1904	276.54812	6.96616	2447826.5805556	27 Oct 1989	1:55:60.00
Radar Nite	527.8062	212.73884	22.42525	2447826.5819444	27 Oct 1989	1:57:60.00
Radar Nite	1110.4456	149.72838	7.27938	2447826.5833333	27 Oct 1989	1:59:60.00
Radar Nite	1360.2813	281.27566	3.29773	2447826.6597222	27 Oct 1989	3:49:60.00
Radar Nite	1052.6357	235.28395	7.84189	2447826.6611111	27 Oct 1989	3:51:60.00
Radar Nite	1500.7202	194.69278	3.08922	2447826.6625000	27 Oct 1989	3:53:60.00
Eye	1886.4860	208.55513	1.10362	2447827.3597222	27 Oct 1989	20:37:60.00
Eye	1042.6527	190.39413	9.95337	2447827.3611111	27 Oct 1989	20:39:60.00
Eye	680.8354	123.13483	16.93138	2447827.3625000	27 Oct 1989	20:41:60.00
Radar Nite	1312.6565	77.08573	3.86010	2447827.3638889	27 Oct 1989	20:43:60.00
Eye	1841.6349	246.99227	0.43637	2447827.4388889	27 Oct 1989	22:31:60.00
Eye	914.5342	237.96272	11.15685	2447827.4402778	27 Oct 1989	22:33:60.00
Radar Nite	373.8777	141.10461	34.85709	2447827.4416667	27 Oct 1989	22:35:60.00
Radar Nite	1152.6843	90.65242	6.09810	2447827.4430556	27 Oct 1989	22:37:60.00
Eye	1027.5726	267.99792	8.15068	2447827.5194444	28 Oct 1989	0:27:60.00
Radar Nite	354.0547	193.22169	37.15572	2447827.5208333	28 Oct 1989	0:29:60.00
Radar Nite	1039.8861	120.66774	8.51512	2447827.5222222	28 Oct 1989	0:31:60.00
Radar Nite	1198.6142	279.81762	5.30112	2447827.5986111	28 Oct 1989	2:21:60.00
Radar Nite	670.7525	224.68996	17.03599	2447827.6000000	28 Oct 1989	2:23:60.00
Radar Nite	1161.3700	166.35389	7.62851	2447827.6013889	28 Oct 1989	2:25:60.00
Radar Nite	1489.1514	282.25136	2.06140	2447827.6777778	28 Oct 1989	4:15:60.00
Radar Nite	1282.9142	241.62790	5.37061	2447827.6791667	28 Oct 1989	4:17:60.00
Radar Nite	1715.6553	207.31945	1.98473	2447827.6805556	28 Oct 1989	4:19:60.00
Eye	1672.9835	220.33454	1.66571	2447828.3777778	28 Oct 1989	21: 3:60.00
Eye	812.2532	199.89954	13.10451	2447828.3791667	28 Oct 1989	21: 5:60.00
Eye	627.2100	110.81914	17.87177	2447828.3805556	28 Oct 1989	21: 7:60.00
Radar Nite	1419.3929	77.37957	2.93510	2447828.3819444	28 Oct 1989	21: 9:60.00
Eye	1682.0727	256.64734	0.75375	2447828.4569444	28 Oct 1989	22:57:60.00
Eye	755.2339	246.17602	13.81707	2447828.4583333	28 Oct 1989	22:59:60.00
Radar Nite	431.7184	124.77448	29.32401	2447828.4597222	28 Oct 1989	23: 1:60.00
Radar Nite	1297.7978	96.46179	5.07336	2447828.4611111	28 Oct 1989	23: 3:60.00
Radar Nite	880.3825	268.53985	10.67522	2447828.5375000	29 Oct 1989	0:53:60.00
Radar Nite	468.2274	174.96352	28.18420	2447828.5388889	29 Oct 1989	0:55:60.00
Radar Nite	1240.7851	130.96218	7.02184	2447828.5402778	29 Oct 1989	0:57:60.00
Radar Nite	1106.6630	273.05680	7.07086	2447828.6166667	29 Oct 1989	2:47:60.00
Radar Nite	881.1470	214.51822	13.07541	2447828.6180556	29 Oct 1989	2:49:60.00
Radar Nite	1480.5495	175.31319	5.24977	2447828.6194444	29 Oct 1989	2:51:60.00
Radar Nite	1549.6968	273.81067	2.29647	2447828.6958333	29 Oct 1989	4:41:60.00
Radar Nite	1610.6864	237.78891	3.14410	2447828.6972222	29 Oct 1989	4:43:60.00

PREDICT TEST Cases Continued

Epoch Data :			Site Data :		Time Data :
2447610.6917756			39.004	Latitude	2447616.50
15.65140042	n	rev/day	-104.883	Longitude	2900.00 min
0.00057550	ndot	rev/2day2	7219.000	Altitude ft	2.00 min
0.0018756	e				
-0.0000489	edot	/day			
51.6244	i				
72.1636	omega	deg			
-5.0693	omega dot	deg/day			
32.7752	argp	deg			
3.7850	argp dot	deg/day			
327.44680	M	deg			

FOR Soviet Space Station MIR

Results:	Range km	Az	El	JD		GMT Time Hr Min Sec
Radar Sun	1557.3661	0.00000	0.00000	2447616.5083333	31 Mar 1989	0:11:60.00
Radar Sun	776.7879	0.00000	0.00000	2447616.5097222	31 Mar 1989	0:13:60.00
Radar Sun	506.6204	0.00000	0.00000	2447616.5111111	31 Mar 1989	0:15:60.00
Radar Sun	1178.9451	0.00000	0.00000	2447616.5125000	31 Mar 1989	0:17:60.00
Radar Sun	1995.5339	0.00000	0.00000	2447616.5138889	31 Mar 1989	0:19:60.00
Eye	2019.2227	287.96247	1.67614	2447616.5750000	31 Mar 1989	1:47:60.00
Eye	1453.7324	310.03127	8.47978	2447616.5763889	31 Mar 1989	1:49:60.00
Eye	1255.2476	347.80567	11.85527	2447616.5777778	31 Mar 1989	1:51:60.00
Eye	1572.0784	22.20823	6.90497	2447616.5791667	31 Mar 1989	1:53:60.00
Eye	2187.1330	40.91777	0.23711	2447616.5805556	31 Mar 1989	1:55:60.00
Eye	1864.8792	324.44752	3.38990	2447616.6430556	31 Mar 1989	3:25:60.00
Eye	1523.1576	352.34685	7.61578	2447616.6444444	31 Mar 1989	3:27:60.00
Radar Nite	1604.4964	25.37638	6.53800	2447616.6458333	31 Mar 1989	3:29:60.00
Radar Nite	2058.2538	49.08307	1.47832	2447616.6472222	31 Mar 1989	3:31:60.00
Radar Nite	1784.0513	325.84264	4.34185	2447616.7097222	31 Mar 1989	5: 1:60.00
Radar Nite	1203.1874	351.09300	13.03254	2447616.7111111	31 Mar 1989	5: 3:60.00
Radar Nite	1072.8630	37.95255	15.94504	2447616.7125000	31 Mar 1989	5: 5:60.00
Radar Nite	1516.2363	72.78209	7.76330	2447616.7138889	31 Mar 1989	5: 7:60.00
Radar Nite	2207.4577	88.46757	0.07867	2447616.7152778	31 Mar 1989	5: 9:60.00
Radar Nite	1529.2055	305.01187	7.58127	2447616.7763889	31 Mar 1989	6:37:60.00
Radar Nite	733.4920	298.41502	27.84168	2447616.7777778	31 Mar 1989	6:39:60.00
Radar Nite	465.0504	158.65741	52.18324	2447616.7791667	31 Mar 1989	6:41:60.00
Radar Nite	1174.0619	137.56062	13.58645	2447616.7805556	31 Mar 1989	6:43:60.00
Radar Nite	2000.9635	134.84919	1.98921	2447616.7819444	31 Mar 1989	6:45:60.00
Radar Nite	2030.5947	268.65142	1.73611	2447616.8430556	31 Mar 1989	8:13:60.00
Radar Nite	1723.6571	243.14890	5.02781	2447616.8444444	31 Mar 1989	8:15:60.00
Radar Nite	1810.7178	214.17797	3.99301	2447616.8458333	31 Mar 1989	8:17:60.00
Radar Sun	1963.8213	0.00000	0.00000	2447617.4638889	31 Mar 1989	23: 7:60.00
Radar Sun	1146.1421	0.00000	0.00000	2447617.4652778	31 Mar 1989	23: 9:60.00
Radar Sun	493.2919	0.00000	0.00000	2447617.4666667	31 Mar 1989	23:11:60.00
Radar Sun	815.0709	0.00000	0.00000	2447617.4680556	31 Mar 1989	23:13:60.00
Radar Sun	1602.8595	0.00000	0.00000	2447617.4694444	31 Mar 1989	23:15:60.00
Radar Sun	2085.3177	0.00000	0.00000	2447617.5305556	1 Apr 1989	0:43:60.00
Radar Sun	1387.8890	0.00000	0.00000	2447617.5319444	1 Apr 1989	0:45:60.00
Radar Sun	970.9358	0.00000	0.00000	2447617.5333333	1 Apr 1989	0:47:60.00
Radar Sun	1188.6644	0.00000	0.00000	2447617.5347222	1 Apr 1989	0:49:60.00
Radar Sun	1823.9628	0.00000	0.00000	2447617.5361111	1 Apr 1989	0:51:60.00
Eye	1961.9481	311.26585	2.32101	2447617.5986111	1 Apr 1989	2:21:60.00
Eye	1548.3600	336.89933	7.20644	2447617.6000000	1 Apr 1989	2:23:60.00
Eye	1535.6211	10.58344	7.41736	2447617.6013889	1 Apr 1989	2:25:60.00
Eye	1931.5906	36.89310	2.71488	2447617.6027778	1 Apr 1989	2:27:60.00
Eye	2123.4635	320.32474	0.80735	2447617.6652778	1 Apr 1989	3:57:60.00
Eye	1550.5917	340.87963	7.23642	2447617.6666667	1 Apr 1989	3:59:60.00
Radar Nite	1309.5397	15.96171	10.99226	2447617.6680556	1 Apr 1989	4: 1:60.00
Radar Nite	1566.2726	50.63020	7.04418	2447617.6694444	1 Apr 1989	4: 3:60.00
Radar Nite	2146.5151	70.72677	0.62403	2447617.6708333	1 Apr 1989	4: 5:60.00
Radar Nite	1929.3774	314.49818	2.74961	2447617.7319444	1 Apr 1989	5:33:60.00
Radar Nite	1134.8566	324.38568	14.45246	2447617.7333333	1 Apr 1989	5:35:60.00
Radar Nite	555.4133	13.60599	40.37159	2447617.7347222	1 Apr 1989	5:37:60.00
Radar Nite	895.3790	95.94735	20.99377	2447617.7361111	1 Apr 1989	5:39:60.00
Radar Nite	1661.2091	110.94454	5.76693	2447617.7375000	1 Apr 1989	5:41:60.00

Epoch Data :			Site Data :		Time Data :
2447616.7557613			39.007	Latitude	2447616.50
15.23989724	n	rev/day	-104.883	Longitude	2900.00 min
0.00472400	ndot	rev/2day2	7219.000	Altitude ft	2.00 min
0.0015500	e				
-0.00041255	edot	/day			
47.6897	i				
17.0000	omega	deg			
-5.16609	omega dot	deg/day			
347.8220	argp	deg			
4.8565	argp dot	deg/day			
12.18930	M	deg			

SDI Test Vehicle

Results:	Range km	Az	El	JD		GMT Time Hr Min Sec
Radar Sun	2491.3441	0.00000	0.00000	2447616.5027778	31 Mar 1989	0: 3:60.00
Radar Sun	1744.4504	0.00000	0.00000	2447616.5041667	31 Mar 1989	0: 5:60.00
Radar Sun	1112.1451	0.00000	0.00000	2447616.5055556	31 Mar 1989	0: 7:60.00
Radar Sun	902.7145	0.00000	0.00000	2447616.5069444	31 Mar 1989	0: 9:60.00
Radar Sun	1334.8551	0.00000	0.00000	2447616.5083333	31 Mar 1989	0:11:60.00
Radar Sun	2029.8286	0.00000	0.00000	2447616.5097222	31 Mar 1989	0:13:60.00
Eye	2436.0022	295.46095	1.34202	2447616.5722222	31 Mar 1989	1:43:60.00
Eye	1751.6968	308.33154	9.31720	2447616.5736111	31 Mar 1989	1:45:60.00
Eye	1244.0800	335.30518	18.97421	2447616.5750000	31 Mar 1989	1:47:60.00
Eye	1181.2137	20.14299	20.71962	2447616.5763889	31 Mar 1989	1:49:60.00
Eye	1616.3785	51.87490	11.55082	2447616.5777778	31 Mar 1989	1:51:60.00
Radar Nite	2275.9654	66.90337	3.10551	2447616.5791667	31 Mar 1989	1:53:60.00
Eye	2118.4894	306.21328	4.74582	2447616.6416667	31 Mar 1989	3:23:60.00
Eye	1377.4916	317.09410	15.97508	2447616.6430556	31 Mar 1989	3:25:60.00
Eye	823.5689	352.37747	35.23764	2447616.6444444	31 Mar 1989	3:27:60.00
Radar Nite	922.0105	64.56409	30.06431	2447616.6458333	31 Mar 1989	3:29:60.00
Radar Nite	1552.6138	89.85455	12.62974	2447616.6472222	31 Mar 1989	3:31:60.00
Radar Nite	2309.8287	98.56740	2.72530	2447616.6486111	31 Mar 1989	3:33:60.00
Eye	2455.2075	295.61743	1.30786	2447616.7097222	31 Mar 1989	5: 1:60.00
Radar Nite	1677.8918	289.28005	10.56793	2447616.7111111	31 Mar 1989	5: 3:60.00
Radar Nite	990.5373	270.71878	27.08861	2447616.7125000	31 Mar 1989	5: 5:60.00
Radar Nite	744.6434	200.51973	40.51695	2447616.7138889	31 Mar 1989	5: 7:60.00
Radar Nite	1241.5301	155.71031	19.05895	2447616.7152778	31 Mar 1989	5: 9:60.00
Radar Nite	1981.1752	144.22351	6.29706	2447616.7166667	31 Mar 1989	5:11:60.00
Radar Nite	2589.0907	269.36919	0.05152	2447616.7791667	31 Mar 1989	6:41:60.00
Radar Nite	2214.0846	250.77124	3.68364	2447616.7805556	31 Mar 1989	6:43:60.00
Radar Nite	2113.5288	227.62076	4.74483	2447616.7819444	31 Mar 1989	6:45:60.00
Radar Nite	2323.9185	205.66616	2.46324	2447616.7833333	31 Mar 1989	6:47:60.00
Radar Sun	2446.8868	0.00000	0.00000	2447617.4861111	31 Mar 1989	23:39:60.00
Radar Sun	1702.6732	0.00000	0.00000	2447617.4875000	31 Mar 1989	23:41:60.00
Radar Sun	1082.3864	0.00000	0.00000	2447617.4888889	31 Mar 1989	23:43:60.00
Radar Sun	911.3134	0.00000	0.00000	2447617.4902778	31 Mar 1989	23:45:60.00
Radar Sun	1370.9522	0.00000	0.00000	2447617.4916667	31 Mar 1989	23:47:60.00
Radar Sun	2073.6509	0.00000	0.00000	2447617.4930556	31 Mar 1989	23:49:60.00
Eye	2370.5525	296.30467	1.89387	2447617.5555556	1 Apr 1989	1:19:60.00
Eye	1694.4411	309.99796	10.03566	2447617.5569444	1 Apr 1989	1:21:60.00
Eye	1213.4324	338.87481	19.55788	2447617.5583333	1 Apr 1989	1:23:60.00
Eye	1202.0828	24.03079	19.91303	2447617.5597222	1 Apr 1989	1:25:60.00
Eye	1669.9637	53.78315	10.52025	2447617.5611111	1 Apr 1989	1:27:60.00
Eye	2341.3572	67.86721	2.30967	2447617.5625000	1 Apr 1989	1:29:60.00
Eye	2023.2694	307.10127	5.72083	2447617.6250000	1 Apr 1989	2:59:60.00
Eye	1291.3953	319.35997	17.68211	2447617.6263889	1 Apr 1989	3: 1:60.00
Eye	786.8104	1.08699	37.09810	2447617.6277778	1 Apr 1989	3: 3:60.00
Radar Nite	982.3397	69.85247	27.09128	2447617.6291667	1 Apr 1989	3: 5:60.00
Radar Nite	1643.3073	91.38986	10.90806	2447617.6305556	1 Apr 1989	3: 7:60.00
Radar Nite	2407.9625	99.26161	1.60714	2447617.6319444	1 Apr 1989	3: 9:60.00
Eye	2331.5258	294.96327	2.39071	2447617.6930556	1 Apr 1989	4:37:60.00
Radar Nite	1559.2440	287.70281	12.30580	2447617.6944444	1 Apr 1989	4:39:60.00
Radar Nite	902.0453	264.66811	30.54213	2447617.6958333	1 Apr 1989	4:41:60.00
Radar Nite	780.3541	188.26426	37.35505	2447617.6972222	1 Apr 1989	4:43:60.00
Radar Nite	1348.9459	152.83354	16.24654	2447617.6986111	1 Apr 1989	4:45:60.00
Radar Nite	2103.6601	143.14627	4.68324	2447617.7000000	1 Apr 1989	4:47:60.00

INTERPLANETARY Test Cases

1.

Altitude at Burnout 200.0 km
Gravitational Parameter Sun 132715440000.0 km3/s2

	Dist from Sun km	Equatorial r km	mu	v	vh1	vh2	vbo	vret	days
Sun	0.0	696000.0	132715440000.0						
Mercury	57900000.0	2440.0	22032.09	47.876	7.5343	9.6137	13.3400	10.5058	105.477
Venus	108100000.0	6051.5	324858.15	35.039	2.5035	2.7160	11.2897	10.6142	145.983
Earth	149599650.0	6378.1	398600.43	29.785					
Mars	227800000.0	3389.9	42828.3	21.857	4.1745	3.5696	11.7735	6.1818	311.804
Jupiter	778000000.0	71492.0	125686537.0	13.061	8.7914	5.6431	14.0882	59.8762	996.945
Saturn	1426000000.0	60268.0	37931187.0	9.647	10.2877	5.4432	15.0674	35.8643	2206.984
Uranus	2868000000.0	25662.0	5793939.0	6.803	11.2799	4.6605	15.7615	22.7246	5849.555
Neptune	4494000000.0	24830.0	6809000.0	5.434	11.6532	4.0549	16.0308	25.2362	11166.384
Pluto	5896000000.0	1150.0	900.0	4.744	11.8129	3.6889	16.1472	10.7638	16587.827

	J2	J3	J4	J6
Sun				
Mercury	0.00008			
Venus				
Earth	0.00108263	-0.254x10-5	-0.161x10-5	
Moon				
Mars	0.001964	0.36x10-4		
Jupiter	0.014736		-0.587x10-3	31x10-6
Saturn	0.016480		-0.936x10-2	
Uranus	0.003349		-3.8x10-5	
Neptune	0.0043			
Pluto				

Bills, Bruce G., Planetary Geodesy, Reviews of Geophysics, Vol 25 No 5 pg 833-839, June 1987.

TU Sun = 54.20765355 days

RENDEZVOUS Test Cases

r 1	r 2	Initial	Revs	Final	Wait Time
6628.1369008	42124.0019008	145.0000000	0	100.7638097	0.8724 TU
6628.1369008	42124.0019008	145.0000000	1	100.7638097	7.9717 TU
6628.1369008	42124.0019008	86.0000000	1	100.7638097	6.8082 TU
6628.1369008	6728.1369008	145.0000000	0	2.0027666	119.0353 TU
6628.1369008	6728.1369008	145.0000000	1	2.0027666	418.7104 TU
6628.1369008	6728.1369008	86.0000000	1	2.0027666	369.5970 TU

Orbit Changes Test Cases

HOHMANN Astro 321 reading

Orbit 1		
Eccentricity	0.0	0.0
Altitude	191.0 km	191.0 km
Orbit 2		
Eccentricity		
Altitude		
Orbit 3		
Eccentricity	0.0	0.0
Altitude	35780.0 km	376310.0 km
DeltaV	0.497806 DU/TU	0.501684 DU/TU
	3.935341 km/s	3.966000 km/s
TOF	23.454289 TU	529.564874 TU
	5.256439 hrs	118.683000 hrs

ONE TANGENT Astro 321 reading

Orbit 1		
Eccentricity	0.0	0.0
Altitude	191.0 km	191.0 km
Orbit 2		
Eccentricity		
Altitude		
True Anomamly	160.0	175.0
Orbit 3		
Eccentricity	0.0	0.0
Altitude	35780.0 km	376310.0 km
DeltaV	0.594466 DU/TU	0.518508 DU/TU
	4.699481 km/s	4.099000 km/s
TOF	15.426199 TU	370.619111 TU
	3.457220 hrs	83.061000 hrs

BI-ELLIPTIC Astro 321 reading

Orbit 1		
Eccentricity	0.0	0.0
Altitude	191.0 km	191.0 km
Orbit 2		
Eccentricity		
Altitude	47836.0 km	503873.0 km (79 DU)
Orbit 3		
Eccentricity		
Altitude	35780.0 km	376310.0 km (59 DU)
DeltaV	0.530399 DU/TU	0.493858 DU/TU
	4.193000 km/s	3.904128 km/s
TOF	86.540700 TU	2650.076000 TU
	19.395000 hrs	593.919630 hrs

REENTRY

Vre = 7200.0 m/s
Flight Path Angle = -45.0 deg
Ballistic Coeff = 4500.0 kg/m2

Alt km	Vel m/s	g's	Alt km	Vel m/s	g's
1.000	2108.397	-54.559	51.000	7190.499	-1.381
2.000	2466.629	-64.996	52.000	7191.712	-1.295
3.000	2828.443	-74.438	53.000	7192.771	-1.220
4.000	3187.101	-82.361	54.000	7193.694	-1.154
5.000	3536.872	-88.418	55.000	7194.499	-1.097
6.000	3873.151	-92.450	56.000	7195.202	-1.048
7.000	4192.466	-94.466	57.000	7195.815	-1.004
8.000	4492.403	-94.606	58.000	7196.349	-0.966
9.000	4771.484	-93.100	59.000	7196.816	-0.933
10.000	5029.025	-90.229	60.000	7197.223	-0.906
11.000	5264.984	-86.290	61.000	7197.577	-0.879
12.000	5479.815	-81.571	62.000	7197.887	-0.857
13.000	5674.342	-76.335	63.000	7198.157	-0.838
14.000	5849.643	-70.811	64.000	7198.392	-0.821
15.000	6006.962	-65.187	65.000	7198.598	-0.807
16.000	6147.630	-59.613	66.000	7198.777	-0.794
17.000	6273.012	-54.203	67.000	7198.933	-0.783
18.000	6384.460	-49.041	68.000	7199.069	-0.773
19.000	6483.283	-44.180	69.000	7199.188	-0.765
20.000	6570.727	-39.655	70.000	7199.292	-0.757
21.000	6647.961	-35.482	71.000	7199.382	-0.751
22.000	6716.067	-31.663	72.000	7199.461	-0.745
23.000	6776.040	-28.192	73.000	7199.530	-0.740
24.000	6828.788	-25.055	74.000	7199.590	-0.736
25.000	6875.131	-22.233	75.000	7199.643	-0.732
26.000	6915.809	-19.706	76.000	7199.688	-0.729
27.000	6951.487	-17.450	77.000	7199.728	-0.726
28.000	6982.756	-15.442	78.000	7199.763	-0.724
29.000	7010.144	-13.660	79.000	7199.793	-0.722
30.000	7034.121	-12.083	80.000	7199.820	-0.720
31.000	7055.101	-10.689	81.000	7199.843	-0.718
32.000	7073.451	-9.459	82.000	7199.863	-0.717
33.000	7089.496	-8.375	83.000	7199.880	-0.716
34.000	7103.520	-7.422	84.000	7199.896	-0.715
35.000	7115.775	-6.584	85.000	7199.909	-0.714
36.000	7126.482	-5.849	86.000	7199.921	-0.713
37.000	7135.834	-5.203	87.000	7199.931	-0.712
38.000	7144.000	-4.638	88.000	7199.940	-0.711
39.000	7151.131	-4.142	89.000	7199.947	-0.711
40.000	7157.357	-3.709	90.000	7199.954	-0.710
41.000	7162.791	-3.329	91.000	7199.960	-0.710
42.000	7167.535	-2.997	92.000	7199.965	-0.710
43.000	7171.675	-2.707	93.000	7199.969	-0.709
44.000	7175.288	-2.453	94.000	7199.973	-0.709
45.000	7178.441	-2.231	95.000	7199.977	-0.709
46.000	7181.192	-2.038	96.000	7199.980	-0.709
47.000	7183.592	-1.868	97.000	7199.982	-0.708
48.000	7185.687	-1.721	98.000	7199.985	-0.708
49.000	7187.514	-1.592	99.000	7199.987	-0.708
50.000	7189.108	-1.479	100.000	7199.988	-0.708

REENTRY Astro 321 problem

Vre = 6504.41 m/s
Flight Path Angle = -27.56 deg
Ballistic Coeff = 1200.00 kg/m2

Alt km	Vel m/s	g's	Alt km	Vel m/s	g's
1.000	5.706	-0.464	51.000	6455.372	-2.498
2.000	14.026	-0.470	52.000	6461.617	-2.241
3.000	30.734	-0.495	53.000	6467.069	-2.017
4.000	60.920	-0.575	54.000	6471.828	-1.820
5.000	110.647	-0.785	55.000	6475.982	-1.648
6.000	186.211	-1.258	56.000	6479.607	-1.498
7.000	293.214	-2.182	57.000	6482.771	-1.367
8.000	435.682	-3.775	58.000	6485.532	-1.252
9.000	615.432	-6.227	59.000	6487.941	-1.151
10.000	831.808	-9.647	60.000	6490.043	-1.064
11.000	1081.802	-14.012	61.000	6491.877	-0.987
12.000	1360.475	-19.154	62.000	6493.477	-0.920
13.000	1661.554	-24.780	63.000	6494.873	-0.862
14.000	1978.083	-30.524	64.000	6496.091	-0.811
15.000	2303.024	-36.004	65.000	6497.153	-0.767
16.000	2629.738	-40.883	66.000	6498.080	-0.728
17.000	2952.330	-44.898	67.000	6498.888	-0.694
18.000	3265.852	-47.890	68.000	6499.594	-0.665
19.000	3566.380	-49.793	69.000	6500.209	-0.639
20.000	3851.002	-50.632	70.000	6500.745	-0.616
21.000	4117.743	-50.494	71.000	6501.214	-0.597
22.000	4365.444	-49.509	72.000	6501.622	-0.580
23.000	4593.630	-47.831	73.000	6501.978	-0.565
24.000	4802.378	-45.619	74.000	6502.289	-0.552
25.000	4992.185	-43.024	75.000	6502.560	-0.540
26.000	5163.854	-40.183	76.000	6502.796	-0.530
27.000	5318.401	-37.213	77.000	6503.002	-0.522
28.000	5456.974	-34.209	78.000	6503.182	-0.514
29.000	5580.786	-31.248	79.000	6503.339	-0.508
30.000	5691.069	-28.386	80.000	6503.476	-0.502
31.000	5789.039	-25.664	81.000	6503.595	-0.497
32.000	5875.867	-23.109	82.000	6503.699	-0.493
33.000	5952.663	-20.735	83.000	6503.790	-0.489
34.000	6020.467	-18.550	84.000	6503.869	-0.485
35.000	6080.237	-16.553	85.000	6503.938	-0.482
36.000	6132.854	-14.741	86.000	6503.999	-0.480
37.000	6179.120	-13.106	87.000	6504.051	-0.478
38.000	6219.759	-11.636	88.000	6504.097	-0.476
39.000	6255.424	-10.320	89.000	6504.137	-0.474
40.000	6286.699	-9.147	90.000	6504.172	-0.473
41.000	6314.105	-8.103	91.000	6504.202	-0.471
42.000	6338.107	-7.178	92.000	6504.229	-0.470
43.000	6359.117	-6.359	93.000	6504.252	-0.469
44.000	6377.499	-5.635	94.000	6504.272	-0.468
45.000	6393.576	-4.997	95.000	6504.290	-0.468
46.000	6407.631	-4.435	96.000	6504.305	-0.467
47.000	6419.916	-3.941	97.000	6504.319	-0.467
48.000	6430.651	-3.507	98.000	6504.330	-0.466
49.000	6440.029	-3.125	99.000	6504.340	-0.466
50.000	6448.219	-2.791	100.000	6504.349	-0.465

HILLS

Start at t=0.0

r = 50.0 0.0 0.0 km
v = 0.0 0.4738 0.0 km/s

Altitude = 180 km

Time min	Position X km	Position Y km	Velocity Required X km/s	Velocity Required Y km/s
0.0000	50.00000	0.00000	Infinite	Infinite
1.0000	49.79160	2.84025	-0.83194	0.06103
2.0000	49.16754	5.66523	-0.41369	0.06079
3.0000	48.13116	8.45976	-0.27335	0.06038
4.0000	46.68804	11.20880	-0.20247	0.05983
5.0000	44.84593	13.89759	-0.15942	0.05912
6.0000	42.61474	16.51165	-0.13031	0.05829
7.0000	40.00647	19.03695	-0.10919	0.05732
8.0000	37.03513	21.45989	-0.09310	0.05625
9.0000	33.71670	23.76747	-0.08039	0.05507
10.0000	30.06903	25.94727	-0.07006	0.05381
11.0000	26.11172	27.98756	-0.06149	0.05248
12.0000	21.86605	29.87739	-0.05426	0.05109
13.0000	17.35484	31.60660	-0.04808	0.04966
14.0000	12.60235	33.16587	-0.04273	0.04819
15.0000	7.63413	34.54684	-0.03808	0.04670
16.0000	2.47688	35.74208	-0.03399	0.04519
17.0000	-2.84166	36.74517	-0.03038	0.04368
18.0000	-8.29290	37.55070	-0.02717	0.04218
19.0000	-13.84754	38.15435	-0.02432	0.04069
20.0000	-19.47570	38.55287	-0.02176	0.03922
21.0000	-25.14715	38.74413	-0.01948	0.03778
22.0000	-30.83137	38.72709	-0.01742	0.03636
23.0000	-36.49782	38.50184	-0.01557	0.03497
24.0000	-42.11602	38.06960	-0.01390	0.03362
25.0000	-47.65578	37.43268	-0.01238	0.03231
26.0000	-53.08731	36.59452	-0.01102	0.03103
27.0000	-58.38141	35.55962	-0.00978	0.02980
28.0000	-63.50962	34.33355	-0.00865	0.02860
29.0000	-68.44436	32.92289	-0.00763	0.02745
30.0000	-73.15911	31.33522	-0.00670	0.02634
31.0000	-77.62852	29.57910	-0.00585	0.02526
32.0000	-81.82856	27.66395	-0.00508	0.02423
33.0000	-85.73665	25.60007	-0.00438	0.02323
34.0000	-89.33178	23.39856	-0.00374	0.02227
35.0000	-92.59463	21.07125	-0.00316	0.02135
36.0000	-95.50764	18.63066	-0.00262	0.02047
37.0000	-98.05516	16.08991	-0.00213	0.01962
38.0000	-100.22349	13.46265	-0.00168	0.01880
39.0000	-102.00098	10.76302	-0.00128	0.01802
40.0000	-103.37806	8.00552	-0.00090	0.01726
41.0000	-104.34734	5.20498	-0.00056	0.01654
42.0000	-104.90361	2.37646	-0.00024	0.01584
43.0000	-105.04387	-0.46484	0.00005	0.01517
44.0000	-104.76737	-3.30364	0.00031	0.01453
45.0000	-104.07559	-6.12469	0.00055	0.01391
46.0000	-102.97226	-8.91279	0.00078	0.01332
47.0000	-101.46331	-11.65298	0.00098	0.01274
48.0000	-99.55685	-14.33052	0.00117	0.01219
49.0000	-97.26312	-16.93102	0.00134	0.01166
50.0000	-94.59447	-19.44050	0.00150	0.01115
51.0000	-91.56523	-21.84545	0.00165	0.01066
52.0000	-88.19170	-24.13297	0.00178	0.01019
53.0000	-84.49201	-26.29073	0.00190	0.00973
54.0000	-80.48605	-28.30716	0.00202	0.00929
55.0000	-76.19536	-30.17139	0.00212	0.00887
56.0000	-71.64300	-31.87342	0.00222	0.00845
57.0000	-66.85345	-33.40409	0.00230	0.00806
58.0000	-61.85247	-34.75517	0.00238	0.00767
59.0000	-56.66693	-35.91940	0.00246	0.00730
60.0000	-51.32471	-36.89052	0.00253	0.00694

HILLS Kaplan pg. 114

Start at t=0.0

r = 50.0   100.0   0.0 km
v = -0.1885 -0.1101 0.0 km/s

Altitude = 222 km

NOTE: This example does not match Kaplan since he uses a YXZ coordinate system, rather than the XYZ coordinate listed here. Using r = 100 x 50 y 0 z will match Kaplan.

	Position		Velocity Required	
Time min	X km	Y km	X km/s	Y km/s
1.0000	50.00000	100.00000	-0.95386	-1.61176
2.0000	39.16109	93.40677	-0.53620	-0.78454
3.0000	29.28783	86.79694	-0.39666	-0.51293
4.0000	20.38005	80.20579	-0.32691	-0.38021
5.0000	12.43247	73.66856	-0.28526	-0.30299
6.0000	5.43463	67.22015	-0.25782	-0.25343
7.0000	-0.62899	60.89503	-0.23861	-0.21959
8.0000	-5.77887	54.72696	-0.22466	-0.19548
9.0000	-10.04041	48.74892	-0.21428	-0.17774
10.0000	-13.44374	42.99282	-0.20645	-0.16436
11.0000	-16.02357	37.48942	-0.20053	-0.15406
12.0000	-17.81902	32.26813	-0.19606	-0.14597
13.0000	-18.87339	27.35682	-0.19273	-0.13950
14.0000	-19.23394	22.78174	-0.19031	-0.13421
15.0000	-18.95165	18.56733	-0.18861	-0.12981
16.0000	-18.08092	14.73610	-0.18751	-0.12606
17.0000	-16.67929	11.30852	-0.18690	-0.12278
18.0000	-14.80715	8.30290	-0.18669	-0.11986
19.0000	-12.52739	5.73530	-0.18678	-0.11719
20.0000	-9.90509	3.61942	-0.18715	-0.11469
21.0000	-7.00716	1.96658	-0.18774	-0.11231
22.0000	-3.90197	0.78560	-0.18850	-0.11001
23.0000	-0.65900	0.08279	-0.18941	-0.10775
24.0000			-0.19042	-0.10550
25.0000			-0.19153	-0.10325
26.0000			-0.19270	-0.10099
27.0000			-0.19392	-0.09870
28.0000			-0.19517	-0.09638
29.0000			-0.19645	-0.09402
30.0000			-0.19774	-0.09162

GROUND TRACK

Time TU	Lat	Lon
0.0000	35.38473	35.66651
0.0996	38.82047	41.15173
0.1992	41.99389	47.04525
0.2989	44.81139	53.35574
0.3985	47.16810	60.06468
0.4981	48.95761	67.11902
0.5977	50.08645	74.42837
0.6974	50.49057	81.86994
0.7970	50.14850	89.30234
0.8966	49.08620	96.58508
0.9962	47.37135	103.59781
1.0958	45.09951	110.25328
1.1955	42.37775	116.50158
1.2951	39.31048	122.32669
1.3947	35.99018	127.73894
1.4943	32.49317	132.76637
1.5940	28.87903	137.44729
1.6936	25.19216	141.82476
1.7932	21.46426	145.94291
1.8928	17.71699	149.84481
1.9924	13.96435	153.57148
2.0921	10.21473	157.16155
2.1917	6.47250	160.65139
2.2913	2.73938	164.07549
2.3909	-0.98450	167.46696
2.4906	-4.69970	170.85810
2.5902	-8.40665	174.28099
2.6898	-12.10479	177.76804
2.7894	-15.79175	-178.64751
2.8890	-19.46230	-174.93116
2.9887	-23.10716	-171.04692
3.0883	-26.71154	-166.95731
3.1879	-30.25345	-162.62390
3.2875	-33.70176	-158.00838
3.3872	-37.01417	-153.07466
3.4868	-40.13562	-147.79221
3.5864	-42.99752	-142.14073
3.6860	-45.51901	-136.11617
3.7856	-47.61109	-129.73716
3.8853	-49.18446	-123.05028
3.9849	-50.16065	-116.13180
4.0845	-50.48443	-109.08385
4.1841	-50.13396	-102.02422
4.2838	-49.12512	-95.07183
4.3834	-47.50828	-88.33190
4.4830	-45.35884	-81.88488
4.5826	-42.76493	-75.78163
4.6822	-39.81587	-70.04441
4.7819	-36.59381	-64.67183
4.8815	-33.16903	-59.64514
4.9811	-29.59837	-54.93440
5.0807	-25.92573	-50.50339
5.1803	-22.18360	-46.31310
5.2800	-18.39508	-42.32394
5.3796	-14.57584	-38.49701
5.4792	-10.73595	-34.79462
5.5788	-6.88135	-31.18029
5.6785	-3.01521	-27.61856
5.7781	0.86099	-24.07454
5.8777	4.74642	-20.51337
5.9773	8.63993	-16.89962
6.0769	12.53902	-13.19675
6.1766	16.43863	-9.36655
6.2762	20.32977	-5.36874
6.3758	24.19776	-1.16092
6.4754	28.02017	3.30118
6.5751	31.76434	8.06255
6.6747	35.38473	13.16672
6.7743	38.82047	18.65194

Orbit Elements

p	=	1.04084
a	=	1.04112
e	=	0.01637
i	=	45.00000
Omega	=	3.07915
Argp	=	50.99729
Nu	=	0.00000
M	=	0.00000
U	=	Undefined
L	=	Undefined
Cappi	=	Undefined

Time

31 Mar 1982 12:00:00.00
or
2445060.0

Cowells Method Results

```
Init 0.000 -0.50960000  0.50960000  0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from     0.000 TU to     6.664 TU
Period        =    6.6639216 TU
Dt            =    0.5000000 TU
Julian Date =  2447538.5000000

RK4t 6.664 -0.52729937  0.48752671  0.71759037 -0.69471172 -0.72206412 -0.01934106
RK4p       -0.52531386  0.48973595  0.71755114 -0.69857574 -0.71828295 -0.02072045
Kep        -0.50960117  0.50959883  0.72068320 -0.70738340 -0.70738500 -0.00000113

 J2 Pert      -0.0011287     0.0011287    -0.0005321       0.0016826
```

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03557220	1.03557748	-4.42762632E-03	-4.422343568E-03
e	0.01999986	0.01966154	0.01965897	-3.38319377E-04	-3.408880625E-04
i	44.99999875	44.99999875	44.99957681	-3.46944695E-18	-4.219406789E-04
Omega	45.00000000	45.00000000	44.61191529	0.00000000E+00	-3.880847054E-01
Argp	90.00000000	90.37079536	90.71715645	3.70795356E-01	7.171564452E-01
Nu	0.00009311	1.21661635	0.97722655	1.21652324E+00	9.771334399E-01
M	0.00008944	1.16947502	0.93936498	1.16938558E+00	9.392755402E-01

```
Init 0.000 -0.50960000  0.50960000  0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from     0.000 TU to     6.664 TU
Period        =    6.6639216 TU
Dt            =    0.1000000 TU
Julian Date =  2447538.5000000

RK4t 6.664 -0.50961330  0.50958551  0.72068236 -0.70737485 -0.70739428 -0.00001374
RK4p       -0.50754460  0.51164801  0.72068130 -0.71110417 -0.70364354 -0.00139344
Kep        -0.50960117  0.50959883  0.72068320 -0.70738340 -0.70738500 -0.00000113

 J2 Pert      -0.0011287     0.0011287    -0.0005321       0.0016826
```

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999847	1.03999848	-1.35829689E-06	-1.350746001E-06
e	0.01999986	0.01999972	0.01999972	-1.38866838E-07	-1.362893335E-07
i	44.99999875	44.99999875	44.99999878	-3.46944695E-18	3.000315298E-08
Omega	45.00000000	45.00000000	44.61804864	1.38777878E-17	-3.819513566E-01
Argp	90.00000000	90.00153899	90.40780034	1.53898621E-03	4.078003448E-01
Nu	0.00009311	359.99956547	359.69916484	3.59999472E+02	3.596990717E+02
M	0.00008944	359.99958260	359.71101989	3.59999493E+02	3.597109304E+02

```
Init 0.000 -0.50960000  0.50960000  0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from     0.000 TU to     6.664 TU
Period        =    6.6639216 TU
Dt            =    0.0500000 TU
Julian Date =  2447538.5000000

RK4t 6.664 -0.50960180  0.50959816  0.72068317 -0.70738295 -0.70738547 -0.00000178
RK4p       -0.50753307  0.51166055  0.72068214 -0.71111218 -0.70363476 -0.00138148
Kep        -0.50960117  0.50959883  0.72068320 -0.70738340 -0.70738500 -0.00000113

 J2 Pert      -0.0011287     0.0011287    -0.0005321       0.0016826
```

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999979	1.03999979	-4.27307209E-08	-3.656743611E-08
e	0.01999986	0.01999985	0.01999986	-4.52759213E-09	-7.739766435E-10
i	44.99999875	44.99999875	44.99999891	1.04083409E-17	1.599299093E-07
Omega	45.00000000	45.00000000	44.61805286	6.93889390E-18	-3.819471414E-01
Argp	90.00000000	90.00010762	90.40640319	1.07624940E-04	4.064031902E-01
Nu	0.00009311	0.00003711	359.69960253	-5.60071466E-05	3.596995094E+02
M	0.00008944	0.00003564	359.71144041	-5.38000397E-05	3.597113510E+02

Cowells Method Results

```
Init 0.000 -0.50960000  0.50960000  0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from     0.000 TU to     6.664 TU
Period       =    6.66392'6 TU
Dt           =    0.0100000 TU
Julian Date =  2447538.5000000

RK4t 6.664 -0.50960119  0.50959881  0.72068320 -0.70738339 -0.70738501 -0.00000115
RK4p       -0.50753246  0.51166120  0.72068217 -0.71111261 -0.70363431 -0.00138085
Kep        -0.50960119  0.50959881  0.72068320 -0.70738339 -0.70738501 -0.00000114

 J2 Pert     -0.0011287      0.0011287      -0.0005321       0.0016826
```

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03999983	-1.36933949E-11	6.100741141E-09
e	0.01999986	0.01999986	0.01999986	-1.46594793E-12	3.782742394E-09
i	44.99999875	44.99999875	44.99999892	3.46944695E-18	1.640886736E-07
Omega	45.00000000	45.00000000	44.61805309	-1.04083409E-17	-3.819469054E-01
Argp	90.00000000	90.00000018	90.40629761	1.76977928E-07	4.062976113E-01
Nu	0.00009459	0.00009448	359.69965807	-1.09209965E-07	3.596995635E+02
M	0.00009087	0.00009076	359.71149377	-1.04906263E-07	3.597114029E+02

```
Init 0.000 -0.50960000  0.50960000  0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from     0.000 TU to     6.664 TU
Period       =    6.6639216 TU
Dt           =    0.0050000 TU
Julian Date =  2447538.5000000

RK4t 6.664 -0.50960119  0.50959881  0.72068320 -0.70738339 -0.70738501 -0.00000115
RK4p       -0.50753246  0.51166120  0.72068217 -0.71111261 -0.70363430 -0.00138085
Kep        -0.50960119  0.50959881  0.72068320 -0.70738339 -0.70738501 -0.00000114

 J2 Pert     -0.0011287      0.0011287      -0.0005321       0.0016826
```

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03999983	-4.28004420E-13	6.113985862E-09
e	0.01999986	0.01999986	0.01999986	-4.58862684E-14	3.784171819E-09
i	44.99999875	44.99999875	44.99999892	0.00000000E+00	1.640898162E-07
Omega	45.00000000	45.00000000	44.61805309	5.89805982E-17	-3.819469051E-01
Argp	90.00000000	90.00000002	90.40629745	1.88659105E-08	4.062974478E-01
Nu	0.00009459	0.00009459	359.69965817	-6.97997142E-09	3.596995636E+02
M	0.00009087	0.00009086	359.71149387	-6.70490757E-09	3.597114030E+02

Cowells Method Results

Init 0.000 -0.50960000 0.50960000 0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from 0.000 TU to 6.664 TU
Period = 6.6639216 TU
Dt = 0.0100000 TU
Julian Date = 2447538.5000000

RK4t 6.664 -0.50960119 0.50959881 0.72068320 -0.70738339 -0.70738501 -0.00000115
RK4p -0.50753246 0.51166120 0.72068217 -0.71111261 -0.70363431 -0.00138085
Kep -0.50960119 0.50959881 0.72068320 -0.70738339 -0.70738501 -0.00000114

J2 Pert -0.0011287 0.0011287 -0.0005321 0.0016826

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03999983	-1.36933949E-11	6.100741141E-09
e	0.01999986	0.01999986	0.01999986	-1.46594793E-12	3.782742394E-09
i	44.99999875	44.99999875	44.99999892	3.46944695E-18	1.640886736E-07
Omega	45.00000000	45.00000000	44.61805309	-1.04083409E-17	-3.819469054E-01
Argp	90.00000000	90.00000018	90.40629761	1.76977928E-07	4.062976113E-01
Nu	0.00009459	0.00009448	359.69965807	-1.09209965E-07	3.596995635E+02
M	0.00009087	0.00009076	359.71149377	-1.04906263E-07	3.597114029E+02

Init 0.000 -0.50960000 0.50960000 0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from 0.000 TU to 6.664 TU
Period = 6.6639216 TU
Dt = 0.0100000 TU
Julian Date = 2447538.5000000

RK4t 6.664 -0.50960119 0.50959881 0.72068320 -0.70738339 -0.70738501 -0.00000115
RK4p -0.50935878 0.50984119 0.72068318 -0.70751242 -0.70725590 0.00018050
Kep -0.50960119 0.50959881 0.72068320 -0.70738339 -0.70738501 -0.00000114

J3 Pert 0.0000010 -0.0000010 -0.0000043 0.0000046

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03999984	-1.36933949E-11	1.047092290E-08
e	0.01999986	0.01999986	0.02000000	-1.46594793E-12	1.361629665E-07
i	44.99999875	44.99999875	44.99999889	3.46944695E-18	1.327148725E-07
Omega	45.00000000	45.00000000	44.99994908	-1.04083409E-17	-5.092395233E-05
Argp	90.00000000	90.00000018	90.21138580	1.76977928E-07	2.113857966E-01
Nu	0.00009459	0.00009448	359.76947415	-1.09209965E-07	3.597693796E+02
M	0.00009087	0.00009076	359.77855866	-1.04906263E-07	3.597784678E+02

Init 0.000 -0.50960000 0.50960000 0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from 0.000 TU to 6.664 TU
Period = 6.6639216 TU
Dt = 0.0100000 TU
Julian Date = 2447538.5000000

RK4t 6.664 -0.50960119 0.50959881 0.72068320 -0.70738339 -0.70738501 -0.00000115
RK4p -0.50961634 0.50958366 0.72068320 -0.70737189 -0.70739651 -0.00001454
Kep -0.50960119 0.50959881 0.72068320 -0.70738339 -0.70738501 -0.00000114

J4 Pert 0.0000010 -0.0000010 -0.0000027 0.0000031

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03999983	-1.36933949E-11	-1.369318222E-11
e	0.01999986	0.01999986	0.01999986	-1.46594793E-12	1.834034599E-13
i	44.99999875	44.99999875	44.99999875	3.46944695E-18	-1.877383665E-12
Omega	45.00000000	45.00000000	45.00016451	-1.04083409E-17	1.645143718E-04
Argp	90.00000000	90.00000018	90.00093124	1.76977928E-07	9.312382997E-04
Nu	0.00009459	0.00009448	0.00025130	-1.09209965E-07	1.567107277E-04
M	0.00009087	0.00009076	0.00024140	-1.04906263E-07	1.505351358E-04

Cowells Method Results

Init 0.000	-0.50960000	0.50960000	0.72068320	-0.70738420	-0.70738420	-0.00000000

Simulate from 0.000 TU to 6.664 TU
Period = 6.6639216 TU
Dt = 0.0100000 TU
Julian Date = 2447538.5000000

RK4t 6.664	-0.50960119	0.50959881	0.72068320	-0.70738339	-0.70738501	-0.00000115
RK4p	-0.50960129	0.50959873	0.72068317	-0.70738337	-0.70738504	-0.00000130
Kep	-0.50960119	0.50959881	0.72068320	-0.70738339	-0.70738501	-0.00000114

Sun	-0.0000000	0.0000000	0.0000000	0.0000001

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03999983	-1.36933949E-11	3.410645269E-11
e	0.01999986	0.01999986	0.01999987	-1.46594793E-12	8.620236502E-09
i	44.99999875	44.99999875	44.99999679	3.46944695E-18	-1.962326295E-06
Omega	45.00000000	45.00000000	44.99999286	-1.04083409E-17	-7.137930274E-06
Argp	90.00000000	90.00000018	90.00003732	1.76977928E-07	3.732088150E-05
Nu	0.00009459	0.00009448	0.00006956	-1.09209965E-07	-2.503655785E-05
M	0.00009087	0.00009076	0.00006682	-1.04906263E-07	-2.404992864E-05

Init 0.000	-0.50960000	0.50960000	0.72068320	-0.70738420	-0.70738420	-0.00000000

Simulate from 0.000 TU to 6.664 TU
Period = 6.6639216 TU
Dt = 0.0100000 TU
Julian Date = 2447538.5000000

RK4t 6.664	-0.50960119	0.50959881	0.72068320	-0.70738339	-0.70738501	-0.00000115
RK4p	-0.50960051	0.50959885	0.72068344	-0.70738306	-0.70738462	-0.00000106
Kep	-0.50960119	0.50959881	0.72068320	-0.70738339	-0.70738501	-0.00000114

Moon	-0.0000002	-0.0000000	-0.0000000	0.0000002

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03999962	-1.36933949E-11	-2.062930837E-07
e	0.01999986	0.01999986	0.01999981	-1.46594793E-12	-4.600062723E-08
i	44.99999875	44.99999875	45.00002620	3.46944695E-18	2.744984133E-05
Omega	45.00000000	45.00000000	44.99997014	-1.04083409E-17	-2.986394181E-05
Argp	90.00000000	90.00000018	90.00003675	1.76977928E-07	3.674635770E-05
Nu	0.00009459	0.00009448	0.00005009	-1.09209965E-07	-4.449951920E-05
M	0.00009087	0.00009076	0.00004812	-1.04906263E-07	-4.274589598E-05

Ballistic Coefficient = 365.8536585 kg/m2

Init 0.000	-0.50960000	0.50960000	0.72068320	-0.70738420	-0.70738420	-0.00000000

Simulate from 0.000 TU to 6.664 TU
Period = 6.6639216 TU
Dt = 0.0100000 TU
Julian Date = 2447538.5000000

RK4t 6.664	-0.50960119	0.50959881	0.72068320	-0.70738339	-0.70738501	-0.00000115
RK4p	-0.51002355	0.50916450	0.72067414	-0.70705869	-0.70764340	-0.00041346
Kep	-0.50960119	0.50959881	0.72068320	-0.70738339	-0.70738501	-0.00000114

Drag	0.0000394	0.0000394	0.0000000	0.0000557

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03987334	-1.36933949E-11	-1.264920157E-04
e	0.01999986	0.01999986	0.01989239	-1.46594793E-12	-1.074718957E-04
i	44.99999875	44.99999875	44.99997419	3.46944695E-18	-2.456034674E-05
Omega	45.00000000	45.00000000	44.99999980	-1.04083409E-17	-1.985707608E-07
Argp	90.00000000	90.00000018	90.00040346	1.76977928E-07	4.034605799E-04
Nu	0.00009459	0.00009448	0.03374512	-1.09209965E-07	3.365052897E-02
M	0.00009087	0.00009076	0.03242235	-1.04906263E-07	3.233148336E-02

Cowells Method Results

```
Init 0.000 -0.50960000  0.50960000  0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from     0.000 TU to     6.664 TU
Period       =    6.6639216 TU
Dt           =    0.0100000 TU
Julian Date =  2447538.5000000

RK4t 6.664 -0.50960119  0.50959681  0.72068320 -0.70738339 -0.70738501 -0.00000115
RK4p       -0.50960122  0.50959879  0.72068320 -0.70738337 -0.70738502 -0.00000116
Kep        -0.50960119  0.50959881  0.72068320 -0.70738339 -0.70738501 -0.00000114

Solar       -0.0000000      0.0000000      0.0000000       0.0000000
```

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03999983	-1.36933949E-11	-1.325367778E-11
e	0.01999986	0.01999986	0.01999986	-1.46594793E-12	-3.204886061E-09
i	44.99999875	44.99999875	44.99999875	3.46944695E-18	-2.539041546E-11
Omega	45.00000000	45.00000000	45.00000001	-1.04083409E-17	5.353852705E-09
Argp	90.00000000	90.00000018	89.99997661	1.76977928E-07	-2.339272480E-05
Nu	0.00009459	0.00009448	0.00012002	-1.09209965E-07	2.542986591E-05
M	0.00009087	0.00009076	0.00011529	-1.04906263E-07	2.442773696E-05

```
Init 0.000 -0.50960000  0.50960000  0.72068320 -0.70738420 -0.70738420 -0.00000000

Simulate from     0.000 TU to     6.664 TU
Period       =    6.6639216 TU
Dt           =    0.0100000 TU
Julian Date =  2447538.5000000

RK4t 6.664 -0.50960119  0.50959881  0.72068320 -0.70738339 -0.70738501 -0.00000115
RK4p       -0.50772080  0.51146328  0.72067363 -0.71091336 -0.70377026 -0.00161726
Kep        -0.50960119  0.50959881  0.72068320 -0.70738339 -0.70738501 -0.00000114

J2 Pert     -0.0011287      0.0011287     -0.0005321       0.0016826
J3 Pert      0.0000010     -0.0000010     -0.0000043       0.0000046
J4 Pert      0.0000010     -0.0000010     -0.0000027       0.0000031
Sun         -0.0000000      0.0000000      0.0000000       0.0000001
Moon        -0.0000002     -0.0000000     -0.0000000       0.0000002
Drag         0.0000394      0.0000394      0.0000000       0.0000557
Solar       -0.0000000      0.0000000      0.0000000       0.0000000
```

	Two Body	RK4 Two Body	RK4 Perturbed	2-Body Delta	Perturbed Delta
a	1.03999983	1.03999983	1.03987702	-1.36933949E-11	-1.228044552E-04
e	0.01999986	0.01999986	0.01989571	-1.46594793E-12	-1.041513196E-04
i	44.99999875	44.99999875	45.00000095	3.46944695E-18	2.202087974E-06
Omega	45.00000000	45.00000000	44.61807138	-1.04083409E-17	-3.819286195E-01
Argp	90.00000000	90.00000018	90.61870002	1.76977928E-07	6.187000175E-01
Nu	0.00009459	0.00009448	359.50259568	-1.09209965E-07	3.595025011E+02
M	0.00009087	0.00009076	359.52209638	-1.04906263E-07	3.595220055E+02
